Sieci Neuronowe

Problemy Współczesnej Nauki i Techniki

Informatyka

Edytor serii: Leonard Bolc

Ryszard Tadeusiewicz

Sieci Neuronowe

Akademicka Oficyna Wydawnicza

Warszawa 1993

Druk i oprawa: Oficyna Wydawnicza READ ME – Drukarnia w Łodzi

Recenzent
Dr Jerzy Cytowski

Redakcja i komputerowy sklad tekstu
Grażyna Domańska-Żurek

Projekt graficzny serii
Akademicka Oficyna Wydawnicza RM

ISBN 83-85769-03-X

Wydanie II. Ark. wyd. 14. Ark. druk. 10.5

Przedmowa

Cala treść prezentowanej książki ma charakter obiektywny i bezosobowy. Opisując na kolejnych stronicach różne typy sieci neuronowych, metody ich uczenia oraz typowe zastosowania oraz metody praktycznej realizacji — staralem się całą uwagę Czytelnika przelać na prezentowane **problemy**, pozostawiając w cieniu ludzi, którzy tworzyli zręby prezentowanej wiedzy i ich pasje, które doporowadziły do opisanych odkryć. Natomiast w tej przedmowie chcialbym — przeciwnie — wspomnieć **o ludziach i ich pasjach**. Bez tych ludzi i bez tych pasji nie bylo by ani wspaniale dziś rozwiniętej dziedziny sieci neuronowych, ani tej książki.

Wiedza o sieciach neuronowych zaczęla się od fascynacji **mózgiem** — narządem, który jeden z klasyków neurobiologii porównal do zimnej owsianki, a którego możliwości przetwarzania informacji wciąż górują nad największymi nawet systemami komputerowymi. Badania procesów przekazywania i przetwarzania informacji w systemie nerwowym czlowieka i zwierząt doprowadziły do nagromadzenia takiej liczby faktów, że ogarnięcie ich wyłącznie wyobraźnią stalo się niewykonalne. Dlatego w latach 40-tych pojawily się próby matematycznego, a potem cybernetycznego opisu funkcjonowania pojedyczych komórek nerwowych i ich zespołów, mających najczęściej formę regularnych sieci. Z nich właśnie wywodzą się dzisiejesze **sieci neuronowe i neurokomputery**.

Liczba ludzi, których prace przyczynily się do powstania i rozwoju neurocybernetyki, jest zbyt wielka, by próbować ich przywoływać. Nazwiska tych największych pojawią się dalej na kartach tej książki, ponieważ w uznaniu ich zasług cale działy nauki o sieciach nerwowych ochrzczono ich nazwiskami: *metody uczenia Widrowa, sieci Kohonena, teoria Grossberga...* Te nazwiska zna dziś każdy, kto zetknął się z problematyką sieci neuronowych. Ja natomiast, korzystając z przywileju jakim jest pisanie tej przedmowy, chcialbym wspomnieć **o wybitnych uczonych — Polakach**, którzy także wnieśli wklad w podstawy teorii sieci neuronowych, ale nie są tak powszechnie znani. Na pierwszym miejscu pozwolę sobie przypomnieć postać **prof. Jerzego Konorskiego**, wybitnego polskiego neurofizjologa, którego pionierskie prace wniosły znaczący wklad do cybernetycznej interpretacji działania systemu nerwowego. Słuchałem Jego wykładów, studiowałem Jego prace, kilkakrotnie rozmawialem z nim w ostatnich latach Jego życia, dlatego zaliczam go do moich Nauczycieli. Zaraz za nim chcialbym wspomnieć **prof. Ryszarda Gawrońskiego**, konstruktora pierwszych polskich sieci neuronowych, którego mialem zaszczyt zaliczać do swoich przyjaciół. Bardzo wysoko cenić trzeba wklad, jaki od zarania badań sieci neuronowych w Polsce wniósł i nadal wnosi **prof. Remigiusz Tarnecki** od strony badań komórek nerwowych i mózgu oraz **prof. Wojciech Zmysłowski** od strony ich modelowania i analiz matematycznych. Wreszcie wspomnieć wypada o polskich badaczach, którzy wnoszą znaczący wklad do rozwoju najnowszej teorii i praktyki sieci neuronowych. Na pierwszym miejscu wymienić tu trzeba **prof. Romana Świniarskiego**, którego prace są szeroko znane na świecie i bardzo wysoko cenione.

Inni badacze, którzy łączą rozmaite podstawowe zainteresowania naukowe (fizyka, elektronika, informatyka) z badaniami sieci neuronowych to między innymi **prof. Włodzisław Duch** z Torunia, **prof. Leszek Rutkowski** z Częstochowy i **prof. Ernest Czogała** z Gliwic.

Oczywiście przedstawiona lista nie jest kompletna i ma zdecydowanie subiektywny charakter — po prostu wymieniłem tych, których osobiście znam, cenię i lubię. Jednak cóż warta była by rola autora, gdyby nie mógł nawet zapewnić swoim przyjaciołom okruchu nieśmiertelności?

Kraków, 1992 Ryszard Tadeusiewicz

Spis treści

Rozdział 1

Wprowadzenie

Technika sieci neuronowych[1] budzi coraz większe zainteresowanie na calym świecie. Istnieją kwartalniki naukowe poświęcone wylącznie tej problematyce (*Neural Networks,* IEEE *Trans. on Neural Networks, Neurocomputing* i inne), a inne renomowane pisma poświęcają sieciom neuronowym cale numery (np. *IEEE Spectrum*) lub cykle artykulów (np. *AI Expert*). Coraz więcej organizuje się konferencji, których przedmiotem są badania w zakresie sieci neuronowych i ich zastosowań, przy czym w Europie ważnym wydarzeniem byly konferencje ECCTD *(European Conference on Circuit Theory)* oraz INNC (*International Neural Network Conference*), ogólnoświatowy zasięg mają — poświęcone w znaczącej części problematyce sieci neuronowych — konferencje organizowane przez IEEE (*The Institute of Electrical and Electronics Engineers, Inc.*) pod nazwą ISCAS (*International Symposium on Circuits and Systems*), zaś w USA na pierwszy plan wysuwa się zdecydowanie cykl konferencji sf IJCNN (*International Joint Conference on Neural Network*) organizowanych przez INNS (*International Neural Network Society*) oraz IEEE. Warto dodać, że ostatnie wymienione Konferencje, gromadzące każdorazowo okolo tysiąca uczestników i ponad pięćset referatów, odbywają się w odstępach **półrocznych** — co stanowi wymowną miarę tempa prac w tej dziedzinie. Na temat badania sieci neuronowych tworzone są już duże międzynarodowe programy badawcze i znaczenie tej problematyki nieprzerwanie rośnie.

Budową i zastosowaniami sieci neuronowych zajął się także (od 1990 roku) slynny amerykański Instytut Inżynierów Elektryków i Elektroników (*Institute of Electrical and Electronics Engineers* w skrócie IEEE). Dwanaście stowarzyszeń wchodzących w sklad tego Instytutu utworzylo wspólnie Radę d/s Sieci Neuronowych (IEEE *Neural Network Council* NNC). Stowarzyszeniami, które weszly w sklad NNC są: *Circuits and Systems, Communications, Control Systems, Engineering in Medicine and Biology, Industry Applications, Industrial Electronics, Information Theory, Lasers and Electro-Optics, Oceanic Engineering, Robotics and Automation, Signal Processing*, oraz *Systems, Man & Cybernetics*. Rada wydaje miesięcznik naukowy IEEE *Transactions on Neural Networks* oraz organizuje dwa razy do roku wspomnianą wyżej konferencję pod nazwą *International Joint Conferences on Neural Networks*.

Problematyka sieci neuronowych jest niewątpliwie dziedziną nauk technicznych. Używając sieci neuronowych jako wygodne systemy przetwarzania informacji, informatycy zapo-

[1]Praca ta powstała w wyniku badań sponsorowanych przez grant KBN nr 42/8/91 (AGH nr 249.123.32).

minają często o korzeniach, czyli o tym, z jakich źródel wywodzi się ta nowoczesna technika. Tymczasem podstawy neurokomputingu oparte są na fundamentalnych odkryciach biologów śledzących tajniki naszego mózgu, odkryciach, z których wiele uhonorowano najwyższym wyrazem naukowego uznania — nagrodą Nobla. Wymieńmy takie ważniejsze odkrycia (podano kolejno nazwiska badaczy, rodzaj odkrycia i datę przyznania Nagrody Nobla):

Pawlow	—	model odruchu warunkowego (1904);
Ramon y Cajal	—	opis struktury sieci nerwowej (1906);
Einthoven	—	rejestracja biopotencjalow tkanek (1924);
Sherrington	—	model nerwowego sterowania mięśni (1932);
Bekesy	—	model percepcji sluchowej (1961);
Hodgkin i Huxley	—	model propagacji sygnalu w aksonie (1963);
Eccles	—	model synapsy (1963);
Granit i Hartline	—	badania mikroelektrodowe (1967);
Katz	—	zasada ,,wszystko albo nic" (1970);
Hubel i Wiesel	—	model kory wzrokowej (1981);
Lorenz i Tinbergen	—	model celowego zachowania (1986).

Warto o tym pamiętać i dlatego przytoczono tu te dane. W książce generalnie będziemy koncentrować uwagę raczej na technicznych aspektach zagadnienia, niż na jego biologicznych ,,korzeniach". Czytelnicy bardziej zainteresowani biocybernetycznym aspektem sieci neuronowych mogą skorzystać z książki [Tade91].

Dla badaczy zajmujących się problemtyką sieci neuronowych w Polsce tematyka ta nie jest zupelnie nieznana. Jeszcze w latach 60-tych popularyzowały ją znakomite prace prof. Ryszarda Gawrońskiego, a i potem sporadycznie pojawiały się prace różnych autorów, poszerzające tę tematykę. Jednak gwaltowny wzrost zainteresowania sieciami neuronowymi, zanotowany pod koniec lat 80-tych i na początku lat 90-tych, zaskoczyl wszystkich i spowodowal wzrost zapotrzebowania na informacje dotyczące tego tematu. Tymczasem aktualne informacje na temat sieci neuronowych nie są ani latwo osiągalne (gdyż są rozsiane po dziesiątkach różnych książek i periodyków) ani tanie. Przykladowo, książki z tego zakresu, jako modne i poszukiwane, osiągają ceny powyżej stu dolarów, (przykladem może tu być uważana przez wielu za podstawę, książka albo dwutomowa ,,biblia" wszystkich badaczy zajmujących się wspólbieżnym przetwarzaniem informacji — praca [Rume90]. Jeszcze wyższe ceny wiążą się z uczestnictwem w specjalistycznych konferencjach, lub nabywaniem (rozprowadzanych przez IEEE) wideokaset z monograficznymi wykladami (podstawowa kaseta zatytulowana *Neural Networks* o oznaczeniu EV0379-8 jest oferowana za 900$). Szczególnie kosztowne są specjalistyczne szkolenia i kursy, chętnie oferowane przez wyspecjalizowane firmy (np. firma Neural-Ware proponuje szkolenie w zakresie stosowania sieci neuronowych w cenie 4095$). Bardzo drogie jest też specjalistyczne oprogramowanie dotyczące modelowania sieci neuronowych (ceny są tu na poziomie kilkunastu tysięcy dolarów, na przyklad znakomity pakiet NeuralWorks firmy NeuralWare kosztuje — zależnie od sposobu skonfigurowania — od 1.500$ do 20.000$), zaś specjalistyczny sprzęt do neurokomputingu kosztuje setki tysięcy dolarów.

Wszystko to sprawia, że zdobywanie wiedzy na temat sieci neuronowych może sprawiać trudności. Ponieważ w trakcie prac badawczych, prowadzonych w ramach wyżej cytowanego ,,grantu" udalo się zgromadzić stosunkowo wiele malo znanych faktów i trudno dostępnych publikacji — postanowiono je skrótowo omówić i wydać drukiem jako swoisty przewodnik problemowy, przydatny dla wszystkich naukowców i praktyków rozpoczynających prace

w zakresie samych sieci neuronowych lub ich licznych zastosowań. Brak podobnej pozycji w polskim piśmiennictwie oraz fakt, że autorowi udalo się zgromadzić stosunkowo dużo danych bibliograficznych z lat 1991 i 1992 — w większości zupelnie nieznanych w Polsce — uzasadnia celowość wydania tej publikacji. Oczywiście przegląd ten nie jest w pelni wyczerpujący. Wiele ważnych pozycji literatury pominięto, gdyż autor — nawet wiedząc o ich istnieniu — nie zdolal do nich dotrzeć. Inne ważne cytaty usunięto świadomie, żeby nie zaciemniać obrazu i nie poszerzać (i tak zbyt dużej) objętości pracy. Tego typu los spotkal między innymi liczne prace na temat sieci neuronowych publikowane przez fizyków. Wreszcie trudno wykluczyć ewentualność, że niektóre ważne prace z tej dziedziny nie są po prostu autorowi znane i zostaly pominięte na skutek tej ignorancji. Niestety, żywiolowy rozwój nauki rodzi także i takie niebezpieczeństwa, od których żaden badacz nie jest nigdy w pelni wolny. Tak więc prezentowany przewodnik problemowy jest — bo musi być — opracowaniem fragmentarycznym, jednak wychodząc z założenia, że od czegoś trzeba zacząć, zdecydowano się go przedstawić w prezentowanej niżej postaci.

Opracowanie podzielono na osiem części. W pierwszej przytoczono kilka podstawowych informacji na temat historii powstania i rozwoju sieci neuronowych. Wydawalo się to potrzebne, ponieważ nie można serio rozwijać żadnej dziedziny nauki nie znając jej korzeni. W drugiej przytoczono wyjaśnienia o charakterze podstawowym i podręcznikowym: co to jest sieć neuronowa, jak jest zbudowana, jak się jej używa itp. W dalszych częściach przedstawiono kolejno dominujące współczesnie typy sieci neuronowych, opisując syntetycznie i **z wykorzystaniem jednolitych oznaczeń i wspólnej terminologii** zarówno budowę tych sieci, jak i sposób ich dzialania oraz zasady uczenia. Książkę uzupelniono obszernym wykazem bibliografii, ulatwiając w ten sposób badaczom wchodzącym dopiero w tę sferę zagadnień orientację w literaturze i wybór własnej problematyki badawczej.

Rozdział 2

Historia powstania i kierunki rozwoju sieci neuronowych

2.1 Dzieje badań nad sieciami neuronowymi

Pierwotnym źródłem badań nad sieciami neuronowymi były prace wielkich klasyków neurofizjologii oraz prace pionierów bioniki. Ich wykaz i omówienie znaleźć można w książce [Tade91]. Jednak można przyjąć, że sama dziedzina sieci neuronowych zaistniała dopiero wraz z wydaniem historycznej pracy [McCu46], w której po raz pierwszy pokuszono się o matematyczny opis komórki nerwowej i powiązanie tego opisu z problemem przetwarzania danych, co rozwinięto w kolejnej pracy tych samych autorów. Prace te miały jednak wymiar wąskiego konkretu, natomiast znacznie ogólniejszą i ambitniejszą podstawę teoretyczną prac nad sieciami neuronowymi stanowiła przez wiele lat para prac: książka [Neum58] ustawiająca tę problematykę z punktu widzenia techniki oraz książka [Tayl60] będąca źródłem wiadomości biologicznych.

Już w tym pionierskim okresie stwierdzono, że najcenniejszą własnością sieci neuronowych jest ich zdolność do przetwarzania informacji w sposób **równoległy**, całkowicie odmienny od szeregowej pracy tradycyjnego komputera. Bardzo szybko stwierdzono także, że zasadniczym atutem sieci może być proces ich uczenia, zastępujący tradycyjne programowanie.

Pierwszym szeroko znanym przykładem zbudowanej i ciekawie działającej sieci neuropodobnej jest *Perceptron* [Rose65]. Był to układ częściowo elektromechaniczny (zmienne wagi synaptyczne realizowano za pomocą potencjometrów obracanych przez odpowiednio sterowane silniki elektryczne), a częściowo elektroniczne (sumowanie pobudzeń). Zbudowany został w 1957 roku przez Franka Rosenblatta i Charlesa Wightmana w Cornell Aeronautical Laboratory. Przeznaczeniem perceptronu było rozpoznawanie znaków alfanumerycznych z procesem uczenia jako metodą programowania systemu. Oto jego podstawowe dane: 8 komórek nerwowych, 512 połączeń (struktura połączeń zadawana była losowo, co wynikało z rozważanej hipotezy badawczej), szybkość działania szacowana była na 10^3 przełączeń na sekundę. Działanie perceptronu nie było zadowalające z punktu widzenia zasadniczego celu badań: układ wprawdzie uczył się rozpoznawania, jednak nie radził sobie z bardziej złożonymi znakami, a ponadto wykazywał wrażliwość na zmianę skali roz-

poznawanych obiektów, ich położenie w polu widzenia (przesunięcie, obrót) oraz zmiany kształtu. Zaletą perceptronu, obok faktu, że był on pierwszą realnie działającą imitacją sieci nerwowej, była zdolność do zachowywania poprawnego działania nawet po uszkodzeniu pewnej części jego elementów. Wraz ze zdolnością do uczenia się oraz wespół z zagadkową na pozór możliwością uzyskania sensownego, celowego działania struktury zbudowanej z elementów połączonych ze sobą w sposób autentycznie losowy — wzrosło ogromnie zainteresowanie eksperymentem perceptronowym w USA i na całym świecie. Powstało niesłychanie dużo naśladownictw, zarówno w Stanach Zjednoczonych, jak i Japonii, Europie i Związku Radzieckim. W tym ostatnim na szczególną uwagę zasługują błyskotliwe prace tragicznie zmarłego Bongarda, w których zasadę perceptronu twórczo uogólniono i zastosowano do identyfikacji między innymi formuł matematycznych. Istniał także (w formie programu symulacyjnego dla komputera Gier) polski perceptron Karpińskiego i Michalskiego (Instytut Automatyki PAN), kilka wersji perceptronów modelował też na komputerach Odra i Cyber autor tej książki wraz ze współpracownikami. Ze względu na ograniczoną objętość tej książki niemożliwe jest nawet skrótowe omówienie tych konstrukcji.

Ciekawą konstrukcją neurokomputera była sieć elektrochemicznych uczących się elementów (nazywanych *Adaline* od Adaptive linear element), zbudowana w 1960 roku przez Bernarda Widrowa z Uniwersytetu Standforda. Sieć ta, nazwana od nazwy elementu *Madaline*, składała się z maksymalnie 8 modeli neuronów o 128 połączeniach i pracowała z szybkością 10^4 przełączeń na sekundę. Jej znaczenie, obok ciekawej zasady działania i niesłychanie efektywnego procesu uczenia, polegało na tym, że był to pierwszy neurokomputer oferowany komercyjnie. Co więcej, układ ten był kupowany i jest używany gdzieniegdzie do dziś, czyli przez ponad 20 lat! Zadania sieci Madaline związane są adaptacyjnym przetwarzaniem sygnałów. Wykorzystywana jest w radarach, sonarach, modemach i liniach telefonicznych.

Opisane wyżej konstrukcje miały bardzo inspirujący wpływ na wielu badaczy, ale nie stanowiły wiernego modelu biologicznej sieci nerwowej, tylko były sztucznie zaprojektowanymi maszynami do przetwarzania informacji, wykorzystującymi elementy neuronowe. Natomiast jedną z najwcześniejszych i do dziś płodnych naukowo prób wiernego odwzorowania w sieci neuronowej obserwowanych w rzeczywistości struktur biologicznych, była budowa sieci z hamowaniem obocznym na wzór struktury systemu wzrokowego kraba *Limulusa*. Sieci te stały się źródłem inspirującej teorii i stanowią do dziś ważny przyczynek do praktyki stosowania sieci neuronowych. W owym pionierskim okresie podejmowano nawet próby tworzenia modeli całego mózgu [Youn68] — wprawdzie tylko ośmiornicy, ale i tak próba ta była dostatecznie zuchwała. Dziś, kiedy wiemy o mózgu znacznie więcej — nikt nie formułuje tak ambitnych celów.

Rozwój badań związanych z problematyką sieci neuronowych został gwałtownie zahamowany na początku lat 70-tych za sprawą książki [Mins69]. Zniechęciła ona wielu badaczy, ponieważ dowodziła, że sieci jednowarstwowe (podobne do perceptronu lub oparte na zasadzie hamowań obocznych) mają bardzo ograniczony zakres zastosowań. Impas, który się wytworzył na blisko 15 lat, przełamała dopiero seria prac pokazujących, że nieliniowe sieci wielowarstwowe wolne są od ograniczeń wskazanych w pracy, a także podających efektywne metody uczenia sieci wielowarstwowych. Nie oznacza to oczywiście, że w latach 70-tych nie było żadnych osiągnięć w dziedzinie sieci neuronowych. Publikowano w tym czasie liczne prace, i to dość znaczące. Jednak po sukcesach, jakimi były niewątpliwie konstrukcje Perceptronu i Madaline, były to osiągnięcia wyraźnie niższego lotu. Dlatego zasadna jest często spotykana opinia, że w latach 70-tych nastąpił ewidentny zastój w badaniach nad modelami

sieci neuronowych. Wynikało to po części ze wspomnianego wyżej fatalnego wpływu książki Minskiego i Paperta, a po części z fascynacji osiągnięciami techniki komputerowej oraz rozwojem technologii układów scalonych bardzo dużych skal integracji. Ówczesnym badaczom wydawało się, że w zakresie typowego przetwarzania informacji nawet najdoskonalsze sieci neuronowe nie będą mogły konkurować z typowymi systemami cyfrowymi, przeto nieliczne prace i badania koncentrowały się na takich zastosowaniach, w których decydujące znaczenie miały możliwości sieci wynikające z jej adaptacyjnego charakteru. Dotyczyło to głównie zagadnień związanych z rozpoznawaniem. Charakterystyczne jest przy tym, że wszystkie konstrukcje lat 70-tych mają już wyłącznie elektroniczną budowę. Oto niektóre z nich:

W 1970 roku Stephen Grossberg buduje na uniwersytecie w Bostonie sieć *Avalanche* służącą do rozpoznawania mowy oraz sterowania ruchami ramienia robota. W tym samym roku w słynnej MIT powstaje *Cerebellatron* Davida Mara, Jamesa Albusa i Andresa Pellioneza, służący także do sterowania robota, a wzorowany nieci na *Avalanche*. W 1974 roku powstaje sieć BP zbudowana przez Paula Werbosa z uniwersytetu Harvarda we współpracy z Davidem Parkeem i Davidem Rumelhartem z uniwersytetu Stanforda. Ta bardzo popularna i łatwo się ucząca sieć wykorzystywana była w podobnym zakresie, jak wyżej wymienione — to znaczy do rozpoznawania i syntezy mowy oraz do sterowania robota. Odmienne zastosowanie miała sieć *Brain State in a Box*, zbudowana przez Jamesa Andersona z uniwersytetu Browna w 1977 roku. Jej zadaniem było wydobywanie wiadomości z bazy danych. Pod względem funkcjonalnym przypominała znane pamięci asocjacyjne z dwustronnym dostępem (BAM), ale jej działanie nie było związane z iteracyjnym procesem poszukiwania rozwiązania, lecz polegało na szybkich zależnościach typu wejście — wyjście.

Opisana wyżej sieć stanowi jednak (pod względem obszaru zastosowań) zdecydowany wyjątek. Absolutną regułą były w tamtym okresie sieci wykorzystywane do rozpoznawania. Taką siecią był *Neocognitron* zbudowany w laboratoriach NHK pod kierunkiem Kunihiko Fukushimy w 1978. Służąc do rozpoznawania ręcznie pisanych znaków. Neocognitron wymagał jednak bardzo wielkiej liczby elementów i połączeń, przeto jest uważany do dziś za najbardziej skomplikowaną sieć neuropodobną, jaką kiedykolwiek zbudowano. Dzięki tej komplikacji mógł jednak odczytywać dowolne znaki (między innymi pismo chińskie), a także był niewrażliwy na zniekształcenia czytanych znaków, na ich rozmiary, przesunięcia, obroty itp. Inna sieć, opracowana w tym samym roku przez Gaila Carpentera i Stephena Grossberga na uniwersytecie w Bostonie, była także przeznaczona do rozpoznawania, a w jej opisach także podkreślano znaczny stopień komplikacji. Sieć nazywała się *Adaptive Resonance Theory* i przystosowana była do rozpoznawania tych sygnałów, których człowiek nie postrzega w sposób bezpośredni: radar, sonar, sygnały wibroakustyczne itp. Sieć ta także odegrała znaczącą rolę w rozwoju neurocyberetyki.

W związku z opracowaniem technologii wytwarzania modeli komórek nerwowych w postaci odpowiednich układów scalonych pojawiły się w latach 80-tych pierwsze konstrukcje o naprawdę dużych rozmiarach i liczącej się mocy obliczeniowej. Najczęściej jednak konstrukcje te wykorzystywały drobny zaledwie fragment możliwości związanych z poznanymi już właściwościami sieci nerwowych. Typową „ofiarą" scalania padała ta własność sieci, którą najtrudniej odwzorować w układzie scalonym — adaptacyjność parametrów.

W tym samym okresie pojawiły się pierwsze sieci ze sprzężeniami zwrotnymi, w których rozwiązywanie stawianych zadań polegało na poszukiwaniu przez sieć stanu równowagi w długim iteracyjnym procesie dynamicznym. Przykładem sieci tego typu są systemy opracowane w 1982 roku przez Johna Hopfielda z AT&T Bell Labs, wykorzystywane do odtwarzania obrazów z ich fragmentów, a także stosowane do rozwiązywania zadań typu optymalizacyj-

nego (np. slynny problem komiwojażera).

Od polowy lat 80-tych notuje się prawdziwy wyścig, którego uczestnikami są, obok laboratoriów badawczych, także firmy produkujące uklady elektroniczne. Osiągnięciami liczącymi się w tym wyścigu są: liczba elementów neuropodobnych, umieszczonych w sieci, liczba połączeń i szybkość działania. Dla zwartości opisu zestawimy ważniejsze elementy tego wyścigu (z okresu pierwszej polowy lat 80-tych) w formie tabelarycznej.

Nazwa	rok	ilość elem.	ilość poł.	szybkość	twórca
Mark III	1985	$8*10^3$	$4*10^5$	$3*10^5$	R.Hercht-Nielsen TRW
Neural emulator Processor	1985	$4*10^3$	$1.6*10^4$	$4.9*10^5$	C.Cruz, IBM
Mark IV	1986	$2.5*10^5$	$5*10^6$	$5*10^6$	R.Hecht-Nielsen TRW
Odyssey	1986	$8*10^3$	$2.5*10^5$	$2*10^6$	A.Penz, Tex. Inst. CRL
Crossbar chip	1986	256	$6.4*10^4$	$6*10^9$	L.Jackel AT&T Bell L.
Anza	1987	$3*10^4$	$5*10^5$	$1.4*10^5$	R.Hecht-Nielsel Neurocomp. Corp.
Parallon	1987	$9.1*10^4$	$3*10^5$	$3*10^4$	S.Bogoch Human Dev.
Anza plus	1988	10^6	$1.5*10^6$	$6*10^6$	R.Hecht-Nielsen Neurocomp. Corp.

Kolejne kolumny tabeli obejmują: nazwę neurokomputera (właśnie w polowie lat 80-tych utarla się ta nazwa), rok opracowania, liczbę elementów neuropodobnych umieszczonych w sieci, liczbę połączeń, szybkość działania (wyrażoną w ilości przełączeń na sekundę) oraz nazwisko twórcy i jego firmę. Większość opracowań ma charakter zbiorowy, przytoczono jednak jedynie pierwsze nazwisko spośród wymienianych w publikacjach twórców.

Jako pewną ciekawostkę, chociaż perespektywicznie o dużym zaczeniu, warto odnotować pojawienie się w latach 80-tych neurokomputerów optycznych. Urządzenia te miały z początku skromne możliwości. Pierwszy z szerzej znanych, opracowany w 1984 przez Dimitrija Psaltisa z Kalifornijskiego Instytutu Technologicznego *Electro-optic crossbar* zawieral zaledwie 32 elementy. Ale już następny z wymienianych, *Optical resonator* zawieral $6.4*10^3$ elementów i $1.6*10^7$ połączeń oraz pracowal z szybkością $1.6*10^5$ przełączeń w ciągu sekundy. Najnowszy z opisywanych w literaturze, zbudowany przez Dana Andersona z uniwersytetu Colorado *Optical novelity filter*, zawiera już $1.6*10^4$ elementów i $2*10^6$ połączeń oraz pracuje z szybkością $2*10^7$ przełączeń w ciągu sekundy.

Na zakończenie warto podkreślić, że zasadnicze znaczenie dla wznowienia zainteresowania sieciami neuronowymi pod koniec lat 80-tych miala książka [Ande88], która spowodowala triumfalny powrót tej tematyki. Bardzo istotny byl tu także ogromny postęp, jaki dokonal się w dziedzinie technologii układów mogących modelować sieci neuronowe oraz rozwój technik ich modelowania. Sytuację końca lat 80-tych i początku lat 90-tych scharakteryzowano ogólnie w trzech kolejnych podrozdzialach.

2.2 Czym jest sieć neuronowa?

Na temat sieci neuronowych napisano sporo prac przeglądowych i popularnych, w tym także sporo krytycznych. Czytelnik potrzebujący popularnego wprowadzenia w samą technikę tych sieci powinien skorzystać z książek wskazanych w załączonej na końcu książki bibliografii, przy czym za szczególnie godną rekomendacji uznać można pracę [Ande88]. Ta książka ma natomiast prezentować tematykę sieci neuronowych w miarę szczegółowo i konkretnie. W kolejnych dalszych rozdzialach przedstawiony zostanie kompletny wyklad najważniejszych zagadnień związanych z budową i wykorzystaniem sieci, wyklad poparty szczegółową formalizacją matematyczną wprowadzanych pojęć i prezentacją nawet drobnych szczegółów dyskutowanych rozwiązań.

Ten rozdzial ma jednak charakter wprowadzający i dlatego przedstawimy na razie dość ogólnie strukturę sieci i ich zastosowania. Rozważania te będą z konieczności skrótowe i bardzo powierzchowne. Niektóre, wybrane wątki przytoczonych tu rozważań zostaną potem obszerniej omówione w dalszych rozdzialach książki. Jednak taki ogólny rzut oka na całość problematyki będzie bardzo przydatny przy dyskutowaniu szczegółów, datego osoby nie zapoznane z techniką sieci neuronowych powinny przestudiować ten rozdzial z należytą uwagą. W literaturze przedmiotu spotkać można rozmaite podejścia. Przedmiotem oddzielnych badań bywa sama pojedyncza komórka nerwowa lub jej fragmenty, zwlaszcza synapsy, akson i dendryty. Sieci neuronowe bywają także samodzielnym obiektem badań teoretycznych jako całość, ponieważ jedną z często wskazywanych przyczyn budowy sieci są próby wyjaśniania przy ich pomocy zasad funkcjonowania naturalnych fragmentów systemu nerwowego na plaszczyźnie neurofizjologii neuroanatomii, ewolucji gatunku i dojrzewania osobniczego, a także psychologii. Prace te zmierzają do interpretacji takich zjawisk, jak homeostaza, inteligencja, uwaga, myślenie koncepcyjne itp. Szczególnie często obiektem takich dociekań bywa generalnie rozumiana percepcja i percepcje konkretnych modalności zmysłowych wraz ze zjawiskami towarzyszącymi, a także ogólne sterowanie zachowaniem w żywych organizmach. Pojawia się nawet nadzieja, że w oparciu o teorię sieci neuronowych uda się zbudować teorię mózgu, chociaż droga do tak zarysowanego celu jest bez wątpienia jeszcze bardzo daleka. Podejmowane są także udane próby wyjaśniania patologii systemu nerwowego poprzez badania modeli sieci neuronowych. Wszystkie te zagadnienia pozostawimy tu bez szerszego komentarza, ponieważ interesują nas tu głównie zastosowania techniczne, a nie implikacje biologiczne. Osoby zainteresowane podanymi aspektami mogą sięgnąć do książki [Tade91].

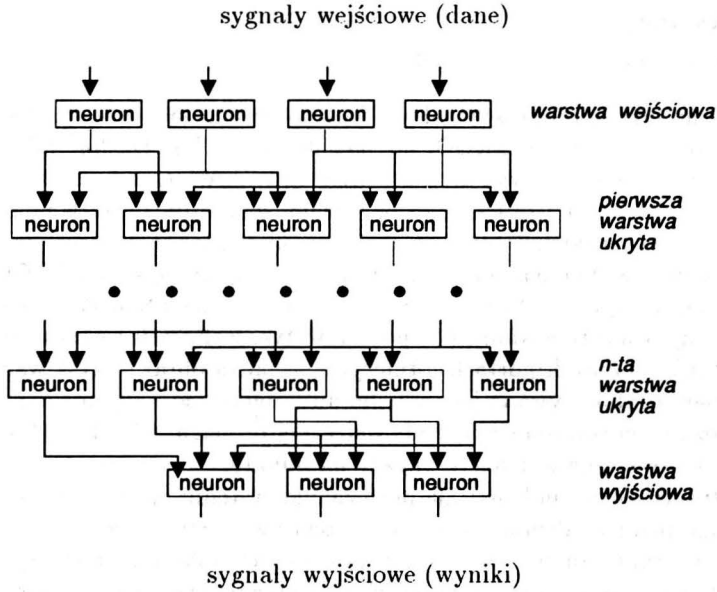
Pierwowzorem wszelkich sieci neuronowych jest oczywiście mózg ludzki, warto więc podać kilka informacji na jego temat. Mózg ma objętość 1400 cm^3 i powierzchnię 2000 cm^2 (kula o tej samej objętości ma zaledwie 600 cm^2!). Masa mózgu wynosi 1,5 kg. Jest to wartość średnia, gdyż są w tym zakresie duże różnice pomiędzy poszczególnymi ludźmi (na przykład mózgi kobiet są z reguły istotnie mniejsze, niż mężczyzn). Nie ma to jednak istotnego znaczenia, gdyż nie istnieje żaden naukowo wykazany związek między wagą mózgu a jego intelektualną sprawnością. Dodajmy zresztą znany fizjologom fakt, że na wymienioną masę mózgu w większości składa się woda ...

Zasadnicze znaczenie dla intelektualnych funkcji mózgu ma pokrywająca półkule kora mózgowa o grubości średnio około 3 mm, zawierająca 10^{10} komórek nerwowych i 10^{12} komórek glejowych. Liczbę połączeń między komórkami szacuje się na 10^{15} przy przeciętnym dystansie od 0,01 mm do 1 m. Komórki nerwowe wysyłają i przyjmują impulsy o częstotliwości 1 — 100 Hz, czasie trwania 1 — 2 ms, napięciu 100 ms i szybkości propagacji

1 — 100 m/s. Szybkość pracy mózgu można szacować na 10 operacji $\times 10^{15}$ synaps $\times$ 100 Hz = 10^{18} operacji/s (dla porównania można podać, że uznawana za najszybszy komputer świata maszyna ETA10 ma szybkość 10^{10} operacji/s). Dla wykonania typowej reakcji mózg realizuje nie więcej niż 100 elementarnych kroków, ponieważ czas reakcji jest nie mniejszy niż 300 ms, a czas refrakcji pojedynczego neuronu wynosi 3 ms. Pojemności informacyjne kanałów zmysłów można szacować jako: wzrok 100 Mb/s, dotyk 1 Mb/s, sluch 15 Kb/s, węch 1 Kb/s, smak 100 b/s.

Sieć neuronowa jest bardzo uproszczonym modelem mózgu. Sklada się ona z dużej liczby (od kilkuset do kilkudziesięciu tysięcy) elementów przetwarzających informację. Elementy te nazywane są neuronami, chociaż w stosunku do rzeczywistych komórek nerwowych ich funkcje są bardzo uproszczone, by nie powiedzieć — sprymitywizowane. Neurony są powiązane w sieć za pomocą połączeń o parametrach (tak zwanych wagach) modyfikowanych w trakcie tak zwanego procesu uczenia. Topologia połączeń oraz ich parametry stanowią program działania sieci, zaś sygnaly pojawiające się na jej wyjściach w odpowiedzi na określne sygnaly wejściowe są rozwiązaniami stawianych jej zadań.

Większość współcześnie budowanych i wykorzystywanych sieci neuronowych ma budowę warstwową, przy czym ze względu na dostępność w trakcie procesu uczenia wyróżnia się warstwy: wejściową, wyjściową oraz tak zwane warstwy ukryte. Dokladniejsze omówienie tych warstw i ich roli przedstawione będzie w dalszych rozdzialach książki.



2.3 Aktualne kierunki badań i zastosowań sieci neuronowych

Trudno wymienić wszystkie aktualnie spotykane zastosowania sieci neuronowych. Magazyn BYTE wymienia między innymi następujące zastosowania:

- diagnostykę ukladów elektronicznych,

- badania psychiatryczne,
- prognozy giełdowe,
- prognozowanie sprzedaży,
- poszukiwania ropy naftowej,
- interpretacja badań biologicznych,
- prognozy cen,
- analiza badań medycznych,
- planowanie remontów maszyn,
- prognozowanie postępów w nauce,
- typowania na wyścigach konnych,
- analiza problemów produkcyjnych,
- optymalizacja działalności handlowej,
- analiza spektralna,
- optymalizacja utylizacji odpadów,
- dobór surowców,
- selekcja celów śledztwa w kryminalistyce,
- dobór pracowników,
- sterowanie procesów przemysłowych.

W tym samym numerze opisano szczegółowo funkcjonowanie systemu opartego na wykorzystaniu sieci neuronowej o nazwie SNOOPE (*System for Nuclear On-line Observation of Potential Explosive*). System ten, wykonywany przez firmę SAIC (producenta neurokomputera Sigma I, procesora Delta II i obszernego oprogramowania dla potrzeb neurokomputingu) służy na wielu amerykańskich lotniskach do kontroli bagażu pasażerów w celu wykrycia ewentualnych ładunków wybuchowych. Pierwszy egzemplarz systemu SNOOPE zainstalowano w międzynarodowych portach lotniczych w Los Angeles i San Francisco w lipcu 1988 roku. Po pomyślnym wypróbowaniu (na ponad 40 tys. walizkach i pakunkach) system ten zainstalowano także w innych portach lotniczych. Analiza obrazu uzyskiwanego w wyniku prześwietlenia bagażu dokonywana jest w pełni automatycznie, za pomocą procesora wykorzystującego możliwości uczenia się metodą wstecznej propagacji błędów (*backpropagation*). Szybkość działania oceniana jest na 10 bagaży na minutę.

Firma *Bendix Aerospace* dokonywała porównania systemu opartego na sieci neuronowej z typowym programem analizującym sygnały sonarowe. Stwierdzono, że sieć wykrywała obiekty podwodne skuteczniej i szybciej, a ponadto charakteryzowała się szybkim uczeniem: jej przystosowanie do nowych zadań wymagało zaledwie kilku godzin w porównaniu z kilkoma miesiącami potrzebnymi do analogicznego przestrojenia typowego programu klasyfikującego.

Firma Nestor zbudowała system automatycznego czytania znaków pisma japońskiego pod nazwą NestorWriter. System ten działa zadowalająco w wielu firmach amerykańskich dzięki zastosowaniu szybkich adaptacyjnych algorytmów rozpoznawania, bazujących na sieciach neuronowych.

Szeroko znany jest program NETtalk dokonujący syntezy mowy na podstawie tekstu pisanego (w języku angielskim). Zastosowane podejście znane jest pod nazwą TDNN (*Time*

Delay Neural Network) i uważane jest za najbardziej efektywną technikę analizy, syntezy, przetwarzania i rozpoznawania sygnałów zależnych od czasu (w tym także sygnału mowy).

Po tych kilku konkretnych przykładach przedstawimy bardziej systemtyczny przegląd zastosowań.

Klasycznym, ale do dziś najczęściej spotykanym obszarem zastosowań technicznych sieci neuronowych są zagadnienia rozpoznawania, a zwlaszcza problemy rozpoznawania kontekstowego i inwariantnego (w sensie niezależności od typowych transformacji). Sieci neuronowe stosuje się także do zadań klasyfikacji oraz do analizy obrazów i ich przetwarzania. Wiele konkretnych prac poświęcono kompresji obrazów i ich segmentacji, problemom odtwarzania oraz „rozumienia" obrazów i wielu podobnym zagadnieniom z pogranicza teorii i praktyki. Prace praktyczne, zwlaszcza w zastosowaniach gospodarczych, dotyczą zwykle zagadnień wąskich i dobrze określonych (na przykład weryfikacja podpisów lub badania uziarnienia surowców mineralnych), ale podejmowane są także zagadnienia ogólniejsze, na przykład rozpoznawnie ręcznie pisanych znaków lub klasyfikacja faktur. Trudno tu oczywiście o dokladne i szczegółowe statystyki, ale z oszacowań autora wynika, że ponad 70% prac dotyczących sieci neuronowych odwołuje się pośrednio lub bezpośrednio do zagadnień rozpoznawania. Jest to tematyka tak silnie dominująca nad pozostalymi zastosowaniami, że niepodobna wyobrazić sobie badacza sieci, który niemialby styczności z systemami wizyjnymi, chociaż zagadnienia analizy i rozpoznawania innych sygnałów także znajdują należne im miejsce — na przykład rozpoznawanie mowy i innych sygnałów dźwiękowych (zwlaszcza w powiązaniu z modelami systemu sluchowego i jego elementów), a także przy analizie tekstów pisanych w języku naturalnym, między innymi w dość „egzotycznym" zakresie ich parafrazowania. Przedmiotem zastosowań sieci neuronowych bywa diagnostyka medyczna, a także analiza sil w elemencie chwytnym robota, sygnałów sonaru lub radaru, sygnałów dotykowych i innych typów sygnałów i informacji.

Pewną egzotyczną, nowo kształtującą się dziedziną zastosowań jest wykorzystanie sieci neuronowych do „klasycznych" zadań przetwarzania sygnałów, takich jak konwersje, filtracje i aproksymacje oraz inne odwzorowania i transformacje (zwlaszcza Fouriera). W literaturze pojawia się coraz więcej prac opisujących struktury takich sieci oraz ich zastosowania.

Inne, bardzo często spotykane, zastosowania sieci neuronowych dotyczą robotyki, automatyki, a także teorii sterowania (zwlaszcza sterowania adaptacyjnego w ukladach samouczących się) i zagadnień optymalizacji. W publikowanych pracach nie unika się także poszczególnych, węższych tematycznie problemów automatyki w rodzaju identyfikacji systemów dynamicznych, sterowania ruchem pojedynczego obiektu (zwykle robota) lub ruchem zbiorowości pojazdów. Wiele prac na temat sieci neuronowych (traktowanych także jako obiekty biologiczne) dotyczy percepcji ruchu i jego planowania, a także koordynacji sensomotorycznej wraz z analizą zależności czasoprzestrzennych. Sieci neuronowe wykorzystuje się także w zagadnieniach pomocniczych, mających związek z automatyką: w metrologii (do oceny błędów sensorów) i w telekomunikacji. Warto dodać,że porównanie zastosowań sieci, jako elementów sterujących, z tradycyjnymi regulatorami wypada zdecydowanie na korzyść sieci.

Z badań nad sieciami neuronowymi najbardziej naturalną korzyść powinna odnieść w niedalekim czasie informatyka. Panuje dość zgodny pogląd, że postęp w dziedzinie sieci neuronowych przybliża zbudowanie pamięci asocjacyjnej (zwlaszcza duże zainteresowanie informatyków budzi klasa tak zwanych dwukierunkowych pamięci asocjacyjnych BAM). Budowa i uczenie sieci neuronowych wnosi też wiele wartościowych przyczynków do problematyki pamięci rozproszonej. Pewnym ubocznym skutkiem tych prac może być też tworzenie co-

raz doskonalszych modeli pamięci biologicznej. Do wieloletniej tradycji należą także ścisle związki łączące problematykę sieci neuronowych z dziedziną tak zwanej sztucznej inteligencji, a zwłaszcza modnych ostatnio systemów ekspertowych. Rozważa się zresztą sieci neuronowe jako generalnie nową technikę obliczeniową, zarówno cyfrową, jak i analogową, oceniając jej efektywność i proponując jej ogólne modele.

Najczęstszym obszarem ogólniejszych badań angażujących sieci neuronowe są prace nad algorytmami automatycznego uczenia się maszyn i ich zdolności do uogólniania nabywanych doświadczeń. Trwają poszukiwania generalnej strategii sterowania uczeniem, podejmowane już w najwcześniejszych pracach [Hebb60], ale ostatnio znacząco rozwinięte i uzupełnione interpretacją geometryczną. Proces uczenia badany jest z punktu widzenia optymalizacji funkcji błędów popełnianych przez sieć, przy czym dokonuje się oceny tego procesu zarówno z punktu widzenia efektywności obliczeniowej, jak i z punktu widzenia czasu wymaganego do osiągnięcia stanu pełnego nauczenia. Zagadnienie to bywa niekiedy rozważane w powiązaniu z tak zwanymi algorytmami genetycznymi, stanowiącymi nową propozycję w zakresie automatyzacji programowania komputerów. Szczególnie ciekawe i trudne zagadnienia uczenia wynikają w kontekście sieci wielowarstwowych, gdzie znaczący postęp uzyskano dopiero w drugiej połowie lat 80-tych [Rume86].

Obecnie rozpatrując uczenie sieci wyróżnia się zwykle uczenie sterowane, szczególnie sugestywnie przedstawione w pracy [Gros90], ale rozważane także w innych publikacjach oraz uczenie „bez nauczyciela" [Widr75], bardziej wiarygodne biologicznie, lecz mające ograniczoną przydatność w technice. Spośród licznych metod uczenia najpowszechniej stosowaną i posiadającą najobszerniejszą literaturę jest metoda wstecznej propagacji błędów, opisana w rozdziale 4, a wykorzystywana obecnie powszechnie, czego dowodem są liczne prace oparte na tej metodzie. Metoda ta jest bardzo skuteczna w praktycznych zastosowaniach, jednak sprawa jej biologicznego prawdopodobieństwa jest wciąż problematyczna.

Sieci neuronowe mają liczne i różnorodne zastosowania poza techniką, na przykład w ekonomii. O wielu z nich będzie jeszcze mowa na kartach tej książki w kontekście omawiania struktury i funkcji konkretnych sieci. Teraz warto jednak wymienić ważniejsze kierunki tych zastosowań bez konkretnej dyskusji o sposobie ich praktycznej realizacji.

1. **Predykcja.** Sieci neuronowe są często wykorzystywane do tego, aby na podstawie pewnych danych wejściowych przewidywać określone dane wyjściowe. Przykładami tego typu zastosowań mogą być: ocena zdolności kredytowej podmiotów gospodarczych, prognozy ekonomiczne rozwoju przedsiębiorstw, prognozy zmian rynku, automatyczne sterowanie obiektów ekonomicznych na podstawie przewidywania (predykcji) ich zachowania w odpowiedzi na sygnały sterujące. Ważną zaletą sieci neuronowych jako urządzeń prognozujących jest fakt, że w wyniku procesu uczenia sieć może nabyć zdolności przewidywania wyjściowych sygnałów (odpowiedzi systemu ekonomicznego) wyłącznie na podstawie obserwacji tak zwanego ciągu uczącego (tj. określonej liczby sekwencji sygnałów wejściowych i wyjściowych stanowiących material empiryczny z przeszłości) *bez konieczności stawiania w sposób jasny hipotez o naturze związku pomiędzy wejściowymi danymi, a przewidywanymi wynikami.* Innymi słowy sieć może nauczyć się prognozować sygnały wyjściowe także wtedy, gdy korzystający z niej badacz nie wie nic o naturze związków łączących przesłanki z wnioskami. Wykorzystano to z powodzeniem do gry na giełdzie, gdzie odpowiednio trenowana sieć neuronowa potrafiła z ponad 70% prawdopodobieństwem wskazywać korzystne lokaty kapitału, prognozować hossę i bessę, przewidywać prawdopodobne kursy akcji na kilka dni naprzód itd.

2. **Klasyfikacja i rozpoznawanie podmiotów gospodarczych.** Zadania klasyfikacji i rozpoznawania, szczególnie często podejmowane i rozwiązywane z wykorzystaniem sieci neuronowych w technice, można rozważać także w zastosowaniach ekonomicznych jako specjalny rodzaj predykcji stanu przedsiębiorstwa. Polega ona na przewidywaniu identyfikatora klasy, do której zaliczyć można wejściowe dane — na przykład pozwala na podstawie danych bilansowych rozpoznać, czy przedsiębiorstwo należy do zwyżkujących gospodarczo, czy przeżywa stagnację lub nawet grozi mu regres. Przykładem rozpoznawania może być zadanie identyfikacji zakładów danej branży, w których warto inwestować lub regionów zagrożonych strukturalnym bezrobociem. Sieć musi nauczyć się znajdować istotne cechy podawanych jej danych całkiem sama, bez wspomagania przez jakąkolwiek z teorii ekonomicznych, co bywa zaletą omawianego tu podejścia ze względu na znaną kruchość teorii ekonomicznych w konfrontacji z rzeczywistymi zagadnieniami praktyki.

3. **Kojarzenie danych.** Dość często kluczem do sukcesu rynkowego i skutecznego zarządzania przedsiębiorstwem jest błyskotliwe kojarzenie różnych faktów. Klasyczne systemy komputerowe są w tym zaresie przeraźliwie indolentne: mogąc gromadzić bardzo wielkie zbiory danych nie dają praktycznie żadnych możliwości ich kojarzenia. Sieci neuronowe dzięki zdolności do uczenia, adaptacji i uogólniania doświadczeń (wszystkie te aspekty będą szczegółowo przedstawione w dalszych rozdziałach) pozwalają zautomatyzować procesy **wnioskowania** na podstawie zgromadzonych danych i ułatwiają menedżerom wykrywanie istotnych powiązań i ważnych aspektów przy praktycznie całkowitym ograniczeniu efektu przytłoczenia nadmiarem szczegółowych informacji, z których trudno wyciągnąć konkretne wnioski.

4. **Analiza danych.** Zadaniem sieci jest w tym wypadku znalezienie związków (być może mających charakter przyczynowy, a być może tylko incydentalnych) występujących w wejściowym zbiorze danych. Realizacja tych zadań przez sieci neuronowe daje zupełnie nowe możliwości w zakresie prowadzenia analiz ekonomicznych i pozwala na wykrycie rzeczywistych intencji podmiotów gospodarczych, co przy agresywnej grze rynkowej i silnej konkurencji bywa kluczem do sukcesu. Analiza danych, prowadzona z wykorzystaniem sieci, pozwala także na ustalenie przyczyn niepowodzeń określonych przedsięwzięć podejmowanych w przeszłości, dzięki czemu łatwiejesze jest unikanie błędów na przyszłość.

5. **Filtracja sygnałów.** Technika ta, rozwinięta głównie w zastosowaniach sieci neuronowych w technice (np. w telekomunikacji czy w automatycznej diagnosyce medycznej), może odegrać bardzo znaczącą rolę przy stosowaniu ich w ekonomii. Wynika to z faktu, że dane gospodarcze dostarczane z różnych źródeł są zakłócone przez liczne subiektywne i obiektywne „szumy informacyjne", w wyniku czego efektywne ich wykorzystanie wymaga wstępnej obróbki tych danych. Klasyczne metody takiej obróbki odwołują się do statystyki, co pozwala eliminować zakłócenia o charakterze losowym, lecz nie daje żadnych podstaw do eliminacji celowych przekłamań lub błędów systematycznych. Klasyczna statystyka nie daje też podstaw do uzupełniania danych niekompletnych, co sieci neuronowe robią szczególnie łatwo. W związku z tym coraz częściej bada się możliwości wykorzystywania sieci neuronowych jako filtrów danych gospodarczych, a doniesienia z form stosujących tę technikę w komputerowym wspomaganiu procesu zarządzania jednoznacznie wskazują na jej zdecydowaną przydatność

i dużą skuteczność.

6. **Optymalizacja.** Sieci neuronowe (zwlaszcza tzw. sieci Hopfielda) doskonale nadają się do poszukiwania rozwiązań prowadzących do optymalnych decyzji gospodarczych. Wykazano doświadczalnie możliwość wykorzystania sieci do rozwiązywania szeregu zadań w zakresie optymalizacji statycznej i dynamicznej, a szczególnie ciekawe są doświadczenia uzyskane podczas korzystania z sieci neuronowych przy rozwiązywaniu zagadnień optymalizacji kombinatorycznej, bardzo trudnych obliczeniowo (czasem np-zupełnych), które z wykorzystaniem sieci mogą być rozwiązywane w krótkim czasie dzięki współbieżnym obliczeniom wykonywanym przez wiele pracujących równocześnie elementów sieci. W jednym z dalszych rozdziałów prezentowana będzie sieć rozwiązująca jedno z klasycznych zadań tego typu — tak zwany problem komiwojażera.

Na tym zastosowania sieci bynajmniej się nie kończą. Sieci neuronowe, same będąc modelami (dalekimi od doskonałości!) naturalnych fragmentów systemu nerwowego człowieka, używane są często jako narzędzie do modelowania innych systemów i zjawisk. Stosunkowo popularne jest na przykład wykorzystanie sieci do tworzenia modeli pewnych zjawisk przestrzennych, jako tak zwanych samoorganizujących się odwzorowań [Koho76]. Rozważane są również systemy podejmowania decyzji i wspomagające wnioskowanie oparte na technice sieci neuronowych na przykład w sieciach energetycznych albo w zarządzaniu liniami metra. Dla tych systemów zaczynają się także pojawiać zastosowania komercyjne głównie w USA i Japonii.

Jak podaje magazyn **Byte** [August 1992] w 1992 roku w USA zakupiono ponad 30.000 pakietów oprogramowania do badań sieci neuronowych. To samo źródło podaje, że w USA działało w 1992 roku ponad 500 firm zajmujących się produkcją sprzętu i oprogramowania dotyczącego sieci neuronowych. Ponieważ w zakresie badań i zastosowań sieci neuronowych utrzymuje się tendencja zwyżkowa — można śmiało przypuszczać, że obecnie firm tworzących neurokomputery, elementy do nich lub oprogramowanie jest ponad pół tysiąca.

Dziedzina ta dostępna jest także dla amatorów, na przykład w USA od kilku lat sprzedawana jest książka [McCl87] zawierająca dyskietkę z programami pozwalającymi na amatorskie modelowanie sieci neuronowych. Zainteresowanie tą książką i związanymi z nią programami stale rośnie.

2.4 Ogólne właściwości sieci neuronowych

Ogromna popularność sieci neuronowych i rosnąca fala ich zastosowań mają swoje konkretne przyczyny. Są one związane z pewnymi właściwościami tych sieci, które warto tutaj skrótowo zestawić i podsumować.

Sieci neuronowe można dzielić według różnych kryteriów. Jednym z ważniejszych jest podział na sieci z jednokierunkowymi połączeniami (nazywanymi często *feedforward*) i sieci ze sprzężeniami zwrotnymi (wiązane często z nazwiskiem Hopfielda). Dynamika sieci ze sprzężeniem zwrotnym dyskutowana jest między innymi w pracy [Hopf82], ale temat ten występuje bardzo często w różnych opracowaniach. W omawianych tu zastosowaniach zagadnienie charakterystyk sieci nie będzie szczegółowo podnoszone, jakkolwiek generalnie można stwierdzić, że sieci ze sprzężeniem zwrotnym chętniej są stosowane w pracach badawczych niż w praktyce.

Jedną z ważniejszych cech sieci jest ich zdolność do adaptacji i samoorganizacji. Jest to atut wykorzystywany w większości niebanalnych zastosowań. Sieci neuronowe cechuje także zmniejszona wrażliwość na uszkodzenia elementów, co częściowo można wyjaśnić na gruncie teorii przytoczonej w pracy [Zmys71]. Wydaje się, że ten aspekt sieci nie zostal jeszcze w pelni wykorzystany w praktyce. Jednak najistotniejszą zaletą sieci neuronowych jest ich zdolność do równoleglej pracy, która warta jest szerszego omówienia.

Systemami technicznymi, z którymi sieci neuronowe najczęściej się porównuje, są systemy wspólbieżne. Jednak stopień wspólbieżności obliczeń jest w sieciach neuronowych setki razy większy, niż w najnowocześniejszych systemach wieloprocesorowych. Powoduje to, że sieci neuronowe dają możliwość znacznego przyspieszenia obliczeń w większości zadań, do których są stosowane. Duże zainteresowanie, jakie budzi zagadnienie sprawności przetwarzania informacji w sieciach neuronowych, jest więc w pelni uzasadnione. Dodatkowym atutem sieci jest wygoda ich programowania poprzez uczenie. Zamiast projektować algorytm wymaganego przetwarzania informacji i dzielić go na moduly nadające się do wspólbieżnego wykonywania — stawia się sieci przykladowe zadania i **automatycznie**, zgodnie z założoną strategią uczenia modyfikuje się połączenia elementów sieci i ich wspólczynniki wagowe. W ten sposób sieć programuje się sama, co czasem prowadzi do rozwiązań w bardzo krótkim czasie, a innym razem może wymagać tysięcy iteracji — ale zawsze przebiega w sposób samoczynny, a więc nie absorbujący dla czlowieka poszukującego określonych rozwiązań.

Często spotyka się zresztą prace łączące i w inny sposób problematykę sieci neuronowych z nowoczesnym podejściem do zagadnień automatyzacji projektowania algorytmów. Rozważa się podejścia polegające na stosowaniu tak zwanego programowania logicznego lub (częściej) zasady samoprogramowania znanej pod nazwą programowania genetycznego. Obok zagadnień automatycznego tworzenia algorymów w sieciach neuronowych w toku procesu uczenia, podejmowane są także tematy reprezentacji danych w sieciach neuronowych oraz ogólnych zasad prowadzenia przez te sieci różnych form obliczeń.

Stosunkowo często są podejmowane ostatnio badania dotyczące dynamiki sieci neuronowych. Wynika z nich konkurencyjność podejścia opartego na sieciach neuronowych w stosunku do klasycznych metod realizacji układów sekwencyjnych. Bardzo inspirujące okazały się także prace wiążące tematykę sieci neuronowych z fizyką statystyczną, a procesy uczenia sieci wiążące z bardzo inspirującą analogią procesów krystalizacji i szkla spinowego. Tematyka ta ma bardzo obszerną bibliografię w czasopismach ściśle fizycznych, którą jednak pominięto w tym zestawieniu ze względu na jego techniczny i aplikacyjny charakter. Czytelników zainteresowanych związkami problematyki sieci nauronowych z zagadnieniami nowoczesnej fizyki wypada odesłać do periodyków fizycznych, z których na szczególne polecenie zasługują: *Physica D, Physical Review* i *J. Statistical Physics* jako pisma, w których problematyka sieci neuronowych gości szczególnie często.

2.5 Narzędzia realizacji sieci neuronowych

2.5.1 Uwagi ogólne

Najczęściej sieci neuronowe są realizowane jako modele matematyczne lub symulacyjne, do czego budowane są niekiedy specjalne komputery (np. procesor DELTA firmy SAIC) albo systemy komputerowe specjalnie dostosowane do wykorzystania w pracach nad sieciami. Badane są także specjalne zasady budowy algorytmów dla sieci neuronowych oraz tworzone jest specjalne oprogramowanie dla potrzeb modelowania sieci, w którym wykorzystuje się

oryginalne podejście do zagadnień modelowania, niekiedy istotnie odmienne od tradycyjnych metod stosowanych przy modelowaniu innych systemów dynamicznych. Warto dodać, że badania sieci neuronowych, które prowadzone są od 1971 roku w Instytucie Automatyki AGH dotyczą właśnie tego aspektu.

W dziedzinie sieci neuronowych z całą pewnością nie powiedziano jeszcze ostatniego słowa. Pojawiają się wciąż nowe, odmienne od wyżej omawianych, koncepcje struktur sieci, prezentowane są nowe pomysły na temat modeli elementów. Tworzone są nowe zastosowania i więcej jest wciąż jeszcze spraw nie rozwiązanych i tematów wartych podjęcia i rozpracowania. Z kolei z faktu, że problematykę sieci neuronowych podejmują wszystkie liczące się ośrodki naukowe na całym świecie wynika, że jest to tematyka niosąca frapujące i obiecujące zagadnienia naukowe, zaś z faktu, że laboratoria badające sieci neuronowe utworzyły wszystkie czołowe firmy zajmujące się elektroniką i informatyką (**Intel, Texas Instruments, IBM, Bell** — by wymienić tylko niektóre ze znanych ośrodków tego typu) wyciągnąć można wniosek, że tematyka ta warta jest wysiłku, gdyż firmy te nie zwykły inwestować w niepewne interesy.

Jest wręcz nieodzowne, aby w tym zbiorowym zainteresowaniu sieciami neuronowymi partycypowali polscy naukowcy. Aby im ułatwić wejście w tę — bardzo już zaawansowaną — tematykę, napisano niniejszą książkę.

Podejmowano wielokrotnie próby określenia, jak wielka jest liczba możliwych do zbudowania sieci. Prace te, oparte zwykle na topologicznym twierdzeniu **Polya** o liczbie grafów jednospójnych dostarczają oszalamiających oszacowań. Jedno z nich podaje na przykład, że już przy 20 neuronach możliwych jest 10^{96} różnych struktur sieci. Inne oszacowanie podaje liczbę rozróżnialnych funkcji, możliwych do realizacji na pojedynczym neuronie. Otóż przy założeniu, że neuron ma 100 wejść liczba tych funkcji osiąga wartość 10^{29}. Nic dziwnego, że przy tak dużej złożoności problem realizacji sieci neuronowych jest trudny i skomplikowany.

Dla rozwoju neurokomputerów niezbędne jest posiadanie sprawnych systemów realizujących sieci neuronowe w sposób hardware'owy. Prace na temat tego typu układów są już bardzo zaawansowane, szczególnie w kontekście wytwarzania specjalizowanych układów scalonych do neurokomputingu, szczególnie w technologii VLSI — uniwersalnych lub specjalnie projektowanych, a także w kontekście elektronicznie scalanych całych systemów, zawierających poza sieciami neuronowymi także inne elementy (np. elektroniczna siatkówka). Prace te mają jednak obecnie raczej badawczy, a nie aplikacyjny charakter, podobnie jak prace dotyczące realizacji funkcji sieci neuronowych za pomocą systemów optycznych. Badaniom tym towarzyszą jednak bardzo ogólne rozważania na temat struktury takich modeli, co doprowadzić może niebawem do prawdziwie rewolucyjnych zmian w sposobie rozumienia procesów przetwarzania informacji w ich najogólniejszym ujęciu. Wylania się także — jako oddzielna dyscyplina naukowa — technika projektowania sieci neuronowych. Nieliczne, ale tym bardziej ciekawe, są systemy komputerowego wspomagania projektowania sieci. Zagadnienia te będą bardziej obszernie dyskutowane w rozdziale 8. Niżej podano syntetyczne informacje na temat oferowanych obecnie handlowych programów i systemów przeznaczonych do wykorzystania techniki sieci neuronowych.

2.5.2 Oprogramowanie do modelowania sieci neuronowych

Obecnie dostępne są stosunkowo liczne (i na ogół dobrej jakości) systemy oferujące oprogramowanie i specjalistyczny sprzęt dla potrzeb badań i rozwoju zastosowań sieci neuronowych. Podany niżej przegląd nie jest kompletny ani wyczerpujący, mimo to może być bardzo

użyteczny przy wyborze własnego podejścia do zagadnień sieci neuronowych. Dostępne są między innymi następujące programy:

ANSkit — zestaw programów pod MS-DOSem (950 $) opartych na MsWindows do projektowania i modelowania sieci(ANS to skrót od *Artificial Neural Sysytems*), współpracuje z **ANSim** pakietem symulacyjnym z wbudowanymi 13 sieciami (495 $) oraz z **ANSpec** — specjalizowanym językiem (2995 $) do modelowani sieci z wykorzystaniem współbieżnej pracy procesora Delta II, Producent **SAIC**.

Awareness — program wprowadzający dla nowicjuszy, pozwalający na wykorzystanie 4 struktur sieci neuronowych. Pracuje w systemie MS-DOS, cena 275 $, producent **Neural Systems**.

AXON — język „drugiej generacji" do opisu i projektowania sieci neuronowych. Pracuje komputerach ANZA. Wspomaga go **Neural Network Development Toolkit**. Produkcja HNC, cena 1950 $ za język i 3950 $ za Toolkit.

BrainMaker — oprogramowanie do symulacji sieci neuronowych pracujące w systemie MS-DOS z szybkością 5 10^5 p/s. Zawiera 5 typów elementów (neuronów). Cena 99.95 $, producent **CSS**.

Cognitron — system do projektowania, modelowania i uruchamiania sieci neuronowych, rozbudowana grafika, możliwość równoleglego uruchomienia wielu sieci, powiązania z Lispem, (wersja **Cognitron Prime** współpracuje z transputerem T800); dostępna wersja dla MS-DOS (600 $) i dla transputera (1800 $), produkcja **Cognitve Software**.

Connections — prosty program demonstrujący rozwiązywanie „problemu komwojażera" za pomocą sieci typu Hopfielda. Pracuje w systemie MS-DOS. Cena 87.95 $, producent **DAIR**.

Delta OS — system operacyjny do sterowania pracą procesora **Delta II** (495 $), z którym współpracuje **Delta II Extended Assembler** (495 $), **Delta ANSpec** (2,995,000 $), **Delta II Power C** (1,995,000 $) i **CARL/BP** — biblioteka programów do uczenia metodą BackPropagation (295 $). Producent **SAIC**.

Desire/Neurnet — system do symulacji sieci neuronowych oparty na języku symulacyjnym Desire, wykorzystującym notację macierzową. Modeluje sieci do 16380 neuronów i dodatkowo pozwala na symulację innych systemów dynamicznych (do 1000 równań różniczkowych), współpracujących z siecią. Szybkość 300.000 połaczeń na sekundę. Wersje /287 (595 $), /387 (695 $), /VAX (1800 $), demo (30 $, z książką 65 $). Producent **Korn**.

DynaMind — pakiet oprogramowania dla IBM PC /AT/386/486 przeznaczony do modelowania sieci i wypracowywania programów implementujących sieci. Wykorzystuje bibliotekę programów w C o nazwie **NeuroLink**, ma wbudowane schematy trzech popularnych sieci. Modeluje sieci o liczbie wejść do 8000 i liczbie neuronów w warstwie do 5000 (limit liczby warstw wynika z dostępnej pamięci PC). Bogata grafika. Cena 145 $ (495 $ w wersji **DynaMind Developer** emulującej chip 80170NX Intela). Producent **NeuroDynamix**.

ExploreNet — pakiet programów modelujący 19 różnych schematów sieci neuronowych z interfejsem **NetSet** i biblioteką modułów współpracujących **UISL** *(User Interface Subroutine Library)*. Sterowany piktogramami i przystosowany do operowania dużymi zbiorami danych (w szczególności do przetwarzania sygnałów). Odmianą programu jest **KnowledgeNet**. Szybkość działania obu programów można zwiększyć za pomocą procesora **Balboa 860**. Cena w wersji MS-DOS 995 $, w wersji VAX/VMS 3950 $, produkcja **HNC**.

ExploreNet 3000 — Nowsza wersja programu **ExploreNet** przystosowana do pracy w systemie **MS Windows 3.0.** Cena 1495 $, producent **HNC**.

Genesis — środowisko programowe do badań nad sieciami neuronowymi. Pracuje w systemie MS-DOS, cena 1095 $, producent **Neural Systems**.

iBrainMaker — wersja programu BrainMaker przeznaczona do symulacji sieci, realizowanych później hardwareowo za pomocą układu scalonego **80170NX** firmy **INTEL**.

iDynaMind — wersja programu DynaMind dostosowana do współpracy z układem neuronowym **80170NX** Intela.

KnowledgeNet — sieć wspomagająca podejmowanie decyzji. Na wejściu oczekiwana jest seria odpowiedzi użytkownika typu tak/nie, zaś na wyjściu podawane są decyzje, ich uzasadnienia, informacje o zanczeniu barkujących danych i stopień wpływu poszczególnych wejściowych informacji na decyzję. Cena 995 $, producent **HNC**.

MacBrain — program do badania i rozwoju sieci neuronowych z wykorzystaniem komputera Macintosh. Możliwości programu rozszerza pakiet (*toolkit*) o nazwie **HyperBrain**, przeznaczony do tworzenia aplikacyjnych wersji sieci w komputerze HyperCube. Cena 995 $, producent **Neurix**.

NeuralBase — specjalistyczna baza danych o najważniejszych osiągnięciach w zakresie neurocybernetyki. Zawiera ponad 2,5 tys.pozycji. Cena 300 $, producent: **ANZA**.

Neural Network Development Tools — oprogramowanie dla projektowania i badania sieci neuronowych. Zawiera m.in. **NeuralWorks Professional II**: Program modelujący 13 typów standardowych sieci neuronowych (z możliwością definicji dalszych, własnych), zawierający 14 regul uczenia, 10 funkcji przejścia i 11 funkcji sumujących. Wersje MS-DOS 1495 $, Sun-4 2995 $. **NeuralWorks Explorer**: program wspomagający wstępne badania. Cena 299 $. **NeuralWork Designer Pack** — pakiet pozwalający projektować i badać sieci (za pomocą pakietu **Professional II**), produkujący po zakończeniu prób źródłowy program w C, będący prototypem użytkowego systemu (a dobrze zdefiniowanym interfejsem użytkownika) zawierającego wymaganą sieć. Cena 1995 $. Producent: **NeuralWare**.

NeuroShell — Program produkujący sieci neuronowe (w postaci programów gotowych do praktycznych zastosowań) drogą uczenia na przykładach przygotowanego programu modelującego sieć. Reklamowany jako alternatywa dla systemów ekspertowych. Funkcjonuje pod systemem MS-DOS. Cena 195 $, producent **Ward**.

Neurosoft — pakiet oprogramowania dla modelowania sieci neuronowych wg. 17 schematów sieci. Przeznaczony dla koprocesorów **ANZA**. Produkcja **HNC**.

N-NET — Zintegrowany pakiet oprogramowania dla projektowania i rozwoju systemów przetwarzania danych opartych na koncepcji sieci neuronowych. Dostępne są wersje dla MS-DOS (895 $) i dla VAX/WMS (2995 $). Produkcja **AI Ware**.

NetWurkz — Program dla systemu MS-DOS stanowiący elementarne wprowadzenie do problematyki sieci neuronowych. Cena 79.95 $, producent **DAIR**.

N1000 — pakiet oprogramowania do tworzenia sieci neuronowych w zastosowaniach do przetwarzania sygnałów i do systemów wizyjnych. Cena 7,955 $, producent **Nestor**.

N500 — Program dla IBM PC modelujący sieć neuronową typu RCE. Cena 495 $, producent **Nestor**.

Owl — biblioteki programów o nazwach **Owl I, Owl II** i **Owl III** przeznaczone dla IBM PC pozwalają na tworzenie i wykorzystanie 10 różnych sieci neuronowych. Cena 349 $. Możliwości pakietu poszerza **Extension Pack** (149 $), który umożliwia definiowanie trzech dalszych schematów sieci. Producent **O&W**.

Savvy — zestaw bibliotek podprogramów (w języku C dla systemu MS-DOS i dla systemu VAX/VMS) wykorzystujących technikę sieci neuronowych do rozwiązywania rzeczywistych problemów. W skład zestawu wchodzą między innymi biblioteki: **Savvy Text Retrival System, Savvy Signal Recognition System, Savvy Vision Recognition System**. Cena do ustalenia z producentem, którym jest **Excalibur**.

The Brain Simulator — oprogramowanie funkcjonujące w systemie MS-DOS, stanowiące połączenie prostego symulatora z podręcznikiem projektowania sieci neuronowych. Cena 99 $, produkcja **AFH**.

2.5.3 Neurokomputery oferowane do sprzedaży

Oferowany jest między innymi następujący sprzęt do zastosowań sieci neuronowych:

ANZA — Koprocesor dla neurokomputingu, tworzący z komputera IBM PC /386 specjalizowaną stację roboczą. Dostępne są dwie wersje dla IBM AT (**ANZA** 7000 $ i **ANZA Plus** 12,500 $) oraz **ANZA Plus/VME** za 24,950 $. Produkcja **ANZA**.

Balboa 860 — koprocesor przystosowany do neurokomputingu, zbudowany w oparciu o procesor **Intel 860 RISC**. Cena 10.950 $, producent **HNC**.

CMAC — procesor neuronowy stosowany jako karta do PC-AT. Nazwa pochodzi od *Cerebellar Model Arithmetic Computer*. Działanie oparte na technice *look-up table*. Wykorzystywany do sterowania robotów. Cena 7950 $, producent **Shenandoach**.

CNAPS — system wieloprocesorowy o architekturze SIMD przeznaczony do obliczeń neuronowych, podłączalny do sieci UNIX Ethernet LAN, reklamowany jako najszybszy na świecie. Nazwa pochodzi od *Connected Network of Adaptive Processors*. Producent **Adaptive Solutions**.

Delta II — procesor na karcie typu IBM PC AT przystosowany do obliczeń sieci neuronowych. 22 MFlops, 12 MBytes, reprezentacja danych 64 bitowa, 1042 rejestry, zewnętrzna magistrala może współpracować do 40 Mbytes/s, 13 modeli sieci 3 warstwowych, producent **SAIC**, cena 24,950 $.

Delta Applications Interface 1016 — 16 buforowanych wejść do procesora **Delta II**, 20 MB/s, zgodne z Data Translation DT Connect, cena 2,000 $, producent **SAIC**.

Dendros-1 — uklad scalony realizujący iloczyn skalarny (22 wagi) będący podstawą modelu neuronu (35 $); można z takich układów budować sieci wykorzytując **Dendros-1 Evaluation Board** (695 $). Producent **Syntonics**.

Hitashi — neurokomputer zawierający (docelowo) ponad 1100 neuronów, wykonany w technologii CMOS (0.8 mikrona), prototyp ma rozmiary 25 x 21 x 20 cm i zawiera 576 układów modelujących neurony. Szybkość 2,3 10^9 p/s. Ceny brak, bo z zasady nie jest sprzedawany poza Japonią. Producent **Hitashi**.

MD/210 Fuzzy Set Comparator — hardware'owa implementacja neuronów Hopfielda. Cena 38 $, producent **Micro Devices**.

iNNTS — system modelujący sieć neuronową na bazie dwóch procesorów **80170NX**, wyposażony w interfejs do IBM PC o nazwie **PCPP** (*Personal Computer Personal Programmer* wstawiany jako karta do PC) i połączony z nim wielożylowym kablem system **GUPI** (*Generic Programmer Interface*), który zawiera 80170 Adaptor oraz modul kontrolny **CTM** (*Confidence Test Module*) służący do programowania (zdawania wag) w układzie **80170NX**. Dla operowania większą niż dwa liczbą układów **80170NX** na raz przewidziano uklad **EMB** (*ETANN Multi-Chip Board*). Cena 8,700 $, producent **Intel**.

Intel 80170NX — chip modelujący (analogowo wewnątrz i w standardzie TTL na zewnątrz) sieć neuronową o 128 elementach. Możliwe różne konfiguracje (feedforward, Hopfield). Określany jako ETANN (*Electrically Trainable Neural Network*) osiąga szybkość 2 10^9 przełączeń na sekundę. Cena 100 $, producent **Intel**.

Intelligent Pattern Recognition Chips — Sieć mająca możliwość pamiętania macierzy wag 1000 x 64 i mnoży je przez wektor wejściowy. Cena 500 $, producent **Oxford Computer**.

NeuroComputer II — stacja robocza zbudowana w oparciu o procesor 68030 (33 MHz), Produkcja **Cognitive Software**.

Neuro-07 — komputer ten (sprzedawany wyłącznie w Japonii w cenie 11 tys $) pracuje z szybkością 216.000 p/s. Producent **NEC**.

NT404NiSP — uklad scalony modelujący sieć neuronową, producent **Neural Technologies**.

NT5000 — neurokomputer przeznaczony do przetwarzania sygnalow (analiza mowy, rozpoznawanie obrazów), wyposażony w interface do kamery i do wejścia akustycznego. Dostarczany wraz z oprogramowaniem (dla PC) pozwalającym wygodnie (edytor graficzny) projektować sieć i ustalać jej parametry; gotowy projekt jest przesylany za pomocą interfejsu do NT5000. Możliwości: 600 neuronów 4500 połączeń (4000 neuronów i 32000 połączeń wersja NT5000T) , wejścia ASCII, analogowe 150 KHz, (CCIR Video Frame Graber — wersja NT5000V), wyjście LCD Display, RS-232, analogowe. Producent **Neural Technologies**.

PowerNet+ — bardzo wydajny symulator sieci neuronowych, pracujący w oparciu o procesor **i860** (64-bitowy o architekturze RISC z zegarem 40 MHz). Pracuje jako karta rozszerzeń do komputera IBM 386/486 i pozwala osiągnąć szybkość pracy rzędu 80 MFLOPS, współpracuje z oprogramowaniem NeuralWorks Professional II firmy NeuralWare. Producent **Scientific Computers Ltd**.

Sigma I Neurocomputer Workstation — IBM PC /386 wraz z procesorem Delta II karta EGA (Sigma II karta VGA), producent SAIC, cena 31,500 $. Dostępne są także: **Sigma II Nuerocomputer Applications Workstation** (39,900 $) i **Sigma III Neurocomputer Applications Workstation** (42,500 $).

TRW Mark V Neural Procesor — wspólbieżny system oparty na procesorze MC 68020 współpracujący z systemem VAX/VMS. Producent **TRW**.

2.5.4 Firmy wytwarzające sprzęt i oprogramowanie dla neurokomputingu

W dziedzinie neurokomputingu działają (między innymi) następujące firmy:

Adaptive Solutions — 1400 N.W. Compton Drive, Suite 340, Beaverton, OR 97006, (fax (503) 690-1249).

AFH — Abbot, Foster & Hauserman, 44 Montgomery, Fifth Floor,San Francisco, CA 94014.

AI Ware — AI Ware, Inc. 1100 Cedar Ave., Suite 212, Cleveland, OH 44106.

ANZA — Anza Research, Inc. 19866 Baywood Drive, Cupertino, CA 95014.

Cognitve Software — Cognitve Software, Inc., 703 East 30th Street, Suite 7, Indianapolis, IN 46205.

CSS — California Scientific Software, 160 East Montecito, Suite E, Sierra Madre, CA 91204.

DAIR — DAIR Computer Systems, 3440 Kenneth Dr., Palo Alto, CA 94303.

Excalibur — Excalibur Technologies, 2300 Buena Vista SE, Albuquerque, NM 87106.

HNC — Hecht-Nielsen Neurocomputers, 5501 Oberlin Drive, San Diego, CA 92121, fax (619) 452-6524.

Korn — G.A. and T.M. Korn Industrial Consultants, 6801 East Opats Street, Tucson, AZ 85715, (602) 298-7054.

Micro Devices — 5695B Beggs Rd., Orlando, FL 32810.

Nestor — Nestor Inc. 1 Richmons Sq., Providence, RI 02906

Neural Systems — 2827 West 43rd Ave., Vancouver, BC Canada, V6N 3H9.

Neural Technologies — 7a Lavant Street, Petersfield, Hampshire, GU32 3EL.

NeuralWare — NeuralWare, Inc. Penn Center West, Building IV, Suite 227, Pittsburgh, PA 15276, fax (412) 787-8220. (Poprzednio: 103 Buckskin Court, Sewickley, PA 15143).

Neurix — Neurix, Inc. 1 Kendall Sq., Suite 2200, Cambridge, MA 02139.

Neuron Data — 156 Univ. Ave., Thrid Floor, Palo Alto, CA 94300.

NeuroDynamix — P.O. Box 323 Boulder, Colorado 80306-0323 USA, tel. 1-800-747-3531.

O&W — Olmsted & Watkins, 2411 East Valley Pkwy., Suite 294, Escondido, CA 92025.

Oxford Computer — 39 Old Good Hill Rd., Oxford, CT 06483.

Perceptis Corp. — 725 Pellissippi Pkwy., Knoxville, TN 37933.

ROSH — ROSH Intelligent Systems Inc., One Needham Place, 50 Cabot St., Needham, MA 02194.

SAIC — Science Applications International Corporation, 10260 Campus Point Drive, MS 71, San Diego, CA 92121.

Scientific Computers Ltd. — Victoria Road, Burgess Hill, West Sussex, UK., RH15 9LW.

Shenandoach — Carl-Henry Piel, Shenandoach Systems Co., 1A Newington Park, West Park Drive, Newington, NH 03801.

Syntonics — Syntonics Systems, Inc., 20790 Northwest Quail Hollow Dr., Portland, OR 97229.

TRW — Military Electronics & Avionics Div., One Rancho Carmel, San Diego, CA 92128.

Ward — Ward Systems Group, Inc., 228 West Patrick St., Frederick, MD 21701.

Przytoczone wyżej dane są niewątpliwie niekompletne, jednak od czegoś przecież trzeba zacząć!

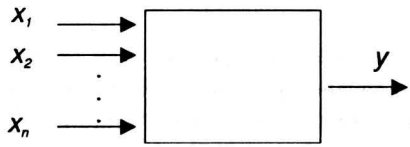
Rozdział 3

Liniowe sieci neuronowe

Prezentację konkretnych sieci neuronowych rozpoczniemy od najprostszego przypadku: sieci liniowych. Sieci takie, znane w literaturze pod nazwami ADALINE i MADALINE, są najprostsze w budowie, a równocześnie są w wielu wypadkach wysoce użyteczne. Dlatego prezentacja tych sieci na początku jest wysoce celowa: Czytelnik pozna na ich przykładzie podstawowe zasady budowy i działania sieci, zyskując przy tym od razu użyteczne narzędzie do zastosowań praktycznych.

3.1 Neurony

Elementy, z których buduje się sieci, charakteryzują się występowaniem wielu wejść i jednego wyjścia.



Sygnały wejściowe $x_i (i = 1, 2, \ldots, n)$ oraz sygnał wyjściowy y mogą przyjmować wartości z pewnego ograniczonego przedziału; z dokładnością do prostej funkcji skalującej można przyjąć, że

$$x_1 \in [-1,\ 1]$$

dla każdego i, a także

$$y \in [-1,\ 1]$$

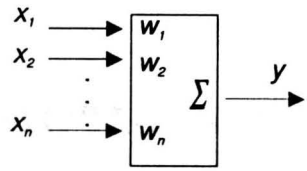
Zależność

$$y = f(x_1, x_2, \ldots, x_n)$$

w najprostszym przypadku może być rozważana jako liniowa:

$$y = \sum_{i=1}^{n} w_i\, x_i$$

przy czym współczynniki w_i, nazywane **wagami synaptycznymi**, mogą podlegać modyfikacjom w trakcie procesu uczenia, który stanowi jeden z zasadniczych wyróżników sieci neuronowych jako adaptacyjnych systemów przetwarzania informacji i o którym będzie dalej mowa w sposób znacznie bardziej szczegółowy. Element opisany przytoczonym wyżej równaniem liniowym jest między inymi zdolny do *rozpoznawania* wejściowych sygnałów.



Aby to wyjaśnić, wystarczy zastosować notację wektorową. Niech zestaw sygnałów wejściowych neuronu tworzy wektor

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

zapisywany przez nas jako kolumna o n składowych. Wektor ten interpretować można także jako punkt w n-wymiarowej przestrzeni $\mathbf{X}$, nazywanej przestrzenią wejść. Wektor ten (i inne wektory) zapisywać niekiedy będziemy w wygodniejszej postaci

$$\mathbf{X} = < x_1, x_2, \ldots, x_n >^T$$

gdzie T jest symbolem transpozycji. Nawiasy $<$ oraz $>$ w powyższym zapisie (a także w innych kontekstach dalej) używane są do grupowania *uporządkowanych* elementów — na przykład składowych wektora. Zestaw n współczynników wagowych także można rozpatrywać jako wektor

$$\mathbf{W} = < w_1, w_2, \ldots, w_n >^T$$

wyznaczający punkt w n-wymiarowej przestrzeni $\mathbf{W}$, nazywanej przestrzenią wag. Przy tych założeniach równanie neuronu wyrazić można jako iloczyn skalarny wektora wejść i wektora wag:

$$y = \mathbf{W} * \mathbf{X}$$

Z formalnego punktu widzenia wygodniej będzie zastąpić iloczyn skalarny (zapisany wyżej z pomocą specjalnego operatora: gwiazdki $*$) zwykłym iloczynem *transponowanego* wektora $\mathbf{W}$ i wektora $\mathbf{X}$

$$y = \mathbf{W}^T \mathbf{X}$$

Z ogólnie znanych właściwości iloczynu skalarnego wynika, że sygnał wyjściowy neuronu y będzie tym większy, im bardziej położenie wektora wejściowego $\mathbf{X}$ w przestrzeni $\mathbf{X}$ przypominać będzie położenie wektora wag $\mathbf{W}$ w przestrzeni $\mathbf{W}$. W ten sposób można powiedzieć,

że neuron rozpoznaje[1] sygnały wejściowe, wyróżniając te, które są podobne do jego wektora wag. Aby to jeszcze silniej zaakcentować załóżmy, że wektory $\mathbf{X}$ i $\mathbf{W}$ są znormalizowane, to znaczy

$$||\mathbf{X}|| = \mathbf{X}^T \mathbf{X} = 1$$

oraz

$$||\mathbf{W}|| = \mathbf{W}^T \mathbf{W} = 1$$

W tym wypadku sygnał wyjściowy neuronu wyznaczyć można ze wzoru

$$y = \cos\varphi$$

gdzie φ jest kątem pomiędzy wektorami $\mathbf{W}$ i $\mathbf{X}$. Jeśli na wejście rozważanego neuronu podawać będziemy różne sygnały $\mathbf{X}$, to wyjście neuronu y będzie miało tym większą wartość, im bardziej podany sygnał $\mathbf{X}$ będzie podobny do „wzorcowego" sygnału, który neuron pamięta w postaci swojego zestawu wag $\mathbf{W}$.

3.2 Warstwa neuronów jako najprostsza sieć

Rozważmy teraz *warstwę* neuronów, z których każdy ma ten sam zestaw sygnałów wejściowych $\mathbf{X} =< x_1, x_2, \ldots, x_n >^T$, natomiast każdy ma swój własny wektor wag. Ponumerujmy neurony w warstwie i oznaczmy jako $\mathbf{W}^{(m)} =< w_1^{(m)}, w_2^{(m)}, \ldots, w_n^{(m)} >^T$ wektor wag m-tego neuronu ($m = 1, 2, \ldots, k$). Wówczas oczywiście sygnał wyjściowy m-tego nauronu można wyznaczyć ze wzoru

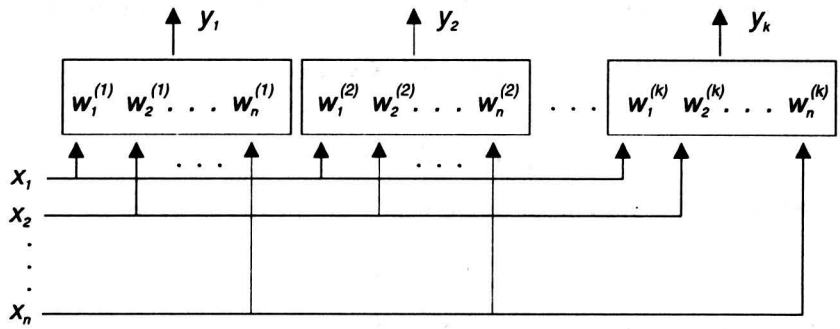
$$y^{(m)} = \mathbf{W}^{(m)} * \mathbf{X} = \sum_{i=1}^{n} w_i^{(m)} x_i$$

Omawiana warstwa stanowi najprostszy przykład sieci neuronowej. Działanie tej sieci polega na tym, że pojawienie się określonego wektora wejściowego X powoduje powstanie sygnałów wyjściowych y_m na wszystkich neuronach wchodzących w skład rozważanej warstwy. Oczekujemy przy tym maksymalnego sygnału wyjściowego y_m na tym neuronie, którego wektor wag $\mathbf{W}^{(m)}$ najbardziej przypomina $\mathbf{X}$. Sieć tego typu może więc rozpoznawać k różnych klas obiektów, gdyż każdy neuron zapamiętuje jeden wzorcowy obiekt, na którego pojawienie się jest „uczulony". O tym, do której klasy należy zaliczyć aktualnie pokazany obiekt, decyduje numer wyjścia, na którym pojawia się sygnał y_m o maksymalnej wartości. Oczywiście „wzorce" poszczególnych klas zawarte są w poszczególnych neuronach w postaci ich wektorów wag $\mathbf{W}^{(m)}$. Neurony dokonujące omówionej tu klasyfikacji sygnałów nazywane są w literaturze *Grandmother Cells*.

Stosując konsekwentnie notację wektorową można sygnały wyjściowe z rozważanej warstwy neuronów zebrać w formie wektora

$$\mathbf{Y} =< y_1, y_2, \ldots, y_k >^T$$

[1] Składowe sygnału wejściowego X można rozważać jako cechy pewnych obiektów, a sygnał wyjściowy y może stanowić miarę podobieństwa tych obiektów do pewnej wyróżnionej klasy. Interpretacja tego typu nawiązuje do często stosowanego modelu *rozpoznawania obrazów* (patrz [Tade91c]).



Wektor ten można wyznaczyć mnożąc wektor wejściowy $\mathbf{X}$ przez macierz $\mathbf{W}_k$ o wymiarach $[k \times n]$, utworzoną w taki sposób, że jej kolejnymi wierszami są (transponowane) kolejne wektory $\mathbf{W}^{(m)}$ (dla $m = 1, 2, \ldots, k$). Macierz $\mathbf{W}_k$ ma więc następującą budowę:

$$\mathbf{W_k} = \begin{bmatrix} w_1^{(1)} & w_2^{(1)} & \ldots & w_n^{(1)} \\ w_1^{(2)} & w_2^{(2)} & \ldots & w_n^{(2)} \\ \vdots & \vdots & & \vdots \\ w_1^{(k)} & w_2^{(k)} & \ldots & w_n^{(k)} \end{bmatrix}$$

Wykorzystując macierz $\mathbf{W}_k$ można zapisać funkcje realizowane przez całą sieć w formie jednego zwartego wzoru:

$$\boldsymbol{Y} = \boldsymbol{W}_k \, \boldsymbol{X}$$

Wzór ten ujawnia jedną z podstawowych własności rozważanej tu sieci neuronowej: Otóż macierz $\mathbf{W}_k$ zadaje określone *odwzorowanie liniowe* sygnału $\mathbf{X} \in \mathcal{R}^n$ w sygnał $\mathbf{Y} \in \mathcal{R}^k$. Odwzorowanie to może być w zasadzie dowolne; jeśli przypomnimy sobie, że znaczna część używanych w teorii przetwarzania sygnałów transformacji jest liniowa (na przykład transformacja Fouriera), wówczas łatwo zauważymy szeroką, praktyczną przydatność rozważanych tu sieci. Celowe jest tu także wprowadzenie jeszcze jednej interpretacji. Przekształcenie sygnału $\mathbf{X}$ w sygnał $\mathbf{Y}$ można traktować jako *filtrację*, w związku z czym o sieci neuronowej dokonującej takiego przekształcenia można mówić jako o *filtrze*. Taka interpretacja jest szczególnie wygodna przy omawianiu sieci MADALINE, o których będzie mowa dalej.

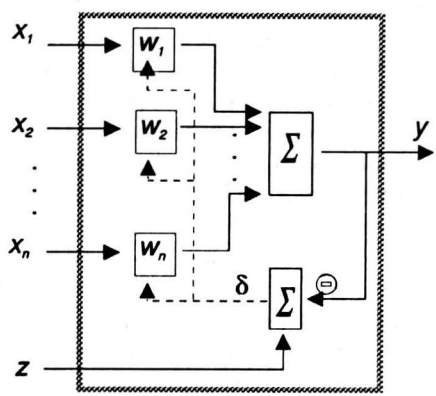
3.3 Uczenie pojedynczego neuronu

O zachowaniu pojedynczego neuronu decydował wektor wag $\mathbf{W}$, a o działaniu sieci — macierz wag $\mathbf{W}_k$. Jak z tego widać wartości wag odgrywają w dziedzinie sieci neuronowych podobną rolę, jak *programy* w dziedzinie obliczeń numerycznych. Jak wiadomo programowanie systemów współbieżnych nastręcza wiele kłopotów, zatem zwykle bardzo trudne

jest aprioryczne ustalenie wektora $\mathbf{W}$ lub macierzy $\mathbf{W}_k$ odpowiednich dla rozwiązywania za pomocą sieci neuronowej jakiegoś złożonego zadania. Na szczęście istnieje możliwość zastąpienia jednorazowego aktu *zaprogramowania* sieci iteracyjnym, wieloetapowym procesem jej *uczenia*. Aby zapewnić możliwość uczenia, trzeba wprowadzony wyżej model neuronu uzupełnić o dwa dodatkowe elementy: procesor zmiany wag i detektor błędu. Tak uzupełniony neuron nazywany bywa ADALINE (*ADAptive LINear Element*) i wykazuje zastanawiająco bogate możliwości w zakresie dostosowywania swojego działania do wymagań wynikających z postawionego zadania. Załóżmy, że zadanie stawiane ADALINE polega na tym, by sygnał wyjściowy y był związany z sygnałami wejściowymi $\mathbf{X}$ pewną zależnością funkcyjną

$$y = f(\mathbf{X})$$

Funkcja f nie musi być zadana w sposób jawny; wystarczy, że dla każdego konkretnego wektora wejściowego potrafimy wskazać konkretną wartość $z = f(\mathbf{X})$ stanowiącą nasze **żądanie** odnośnie sygnału wyjściowego y.



Zasada działania ADALINE przy rozwiązywaniu tego zadania oparta jest na podstawowym algorytmie uczenia, wprowadzonym przez Widrowa i Hoffa [Widr60]. Algorytm ten, nazywany regułą DELTA, zakłada, że wraz z każdym wektorem wejściowym $\mathbf{X}$ do neuronu podawany jest sygnał z, opisany wyżej jako *zadana* (wymagana) odpowiedź nauronu na sygnał $\mathbf{X}$. Neuron odpowiada na sygnał $\mathbf{X}$ sygnałem $y = \mathbf{W} * \mathbf{X}$, przy czym jeśli neuron nie jest nauczony, sygnał ten jest inny, niż wymagany ($y \neq z$). Wewnątrz neuronu ADALINE istnieje blok oceniający wielkość błędu

$$\delta = z - y$$

Blok ten składa się z inwertora (dla uzyskania z sygnału y sygnału $-y$) oraz sumatora. Na schemacie, funkcję inwertora zasygnalizowano symbolem $\ominus$ przy odpowiednim wejściu sumatora. Na podstawie sygnału błędu δ oraz wektora wejściowego $\mathbf{X}$ możliwe jest takie

skorygowanie wektora wag $\mathbf{W}$, by neuron lepiej realizował zadaną funkcję $y = f(\mathbf{X})$. Nowy wektor wag $\mathbf{W}'$ obliczany jest ze wzoru

$$\mathbf{W}' = \mathbf{W} + \eta\,\delta\,\mathbf{X}$$

gdzie η jest współczynnikiem liczbowym, decydującym o szybkości uczenia. Zasadom wyboru tego współczynnika poświęcimy nieco uwagi później.

Za chwilę dokonamy matematycznego uzasadnienia, że podana metoda uczenia działa poprawnie. Zanim to jednak nastąpi — warto zastanowić się nad sensem podanej metody uczenia i spróbować intuicyjnie uzmysłowić sobie, dlaczego możemy oczekiwać, że metoda ta da dobre wyniki. Zacznijmy od interpretacji geometrycznej. Załóżmy, że $\delta > 0$, to znaczy, że $z > y$. Oznacza to, że sygnał wyjściowy z neuronu był za *mały*. Ponieważ sygnał ten zależy od kąta między wektorami $\mathbf{X}$ i $\mathbf{W}$ — domyślamy się, że kąt ten był za duży. Żeby sygnał y był maksymalny — trzeba by było uzgodnić kierunki wektorów $\mathbf{X}$ i $\mathbf{W}$. Dodając (wektorowo) do $\mathbf{W}$ wektor $\mathbf{X}$, otrzymujemy nowy wektor $\mathbf{W}' = \mathbf{W} + \mathbf{X}$ w każdym wypadku bliższy $\mathbf{X}$ niż poprzedni wektor $\mathbf{W}$. W istocie reguła uczenia nakazuje dodawanie *fragmentu* wektora $\mathbf{X}$ (ponieważ zwykle $\eta\,\delta < 1$), co jest uzasadnione, gdyż zapobiega zbyt gwałtownym „obrotom" wektora $\mathbf{W}$ przy każdym napotkanym błędzie. Bardzo podobne rozumowanie pozwala się upewnić, że przy $\delta < 0$ następuje oddalanie wektora $\mathbf{W}$ od wektora $\mathbf{X}$, który wywołał błąd polegający na zbyt silnej odpowiedzi neuronu ($y > z$).

Inne rozumowanie, które uwiarygodnia opisaną regułę uczenia, opiera się na spostrzeżeniu, że korekta wektora $\mathbf{W}$ jest tym silniejsza, im większy został odnotowany błąd (wielkość korekty jest proporcjonalna do δ). Jest to uzasadnione: trzeba silnie interweniować przy dużych błędach i dokonywać płynnego, subtelnego dostrajania w przypadku błędów małych. Przy okazji zapewniony jest automatycznie logicznie konieczny warunek, że w przypadku braku błędu ($\delta = 0$) żadne korekty nie będą dokonywane. Kolejną zgodną ze zdrowym rozsądkiem zasadę dostrzeżemy w opisanej regule uczenia skupiając uwagę na pojedynczych składowych wektorów $\mathbf{W}$ i $\mathbf{X}$. Widać, że i-ta składowa wektora $\mathbf{W}$ będzie tym silniej zmieniona w wyniku procesu uczenia, im większa była odpowiadająca jej składowa wektora $\mathbf{X}$. Jest to prawidłowe i logiczne: jeśli odpowiednia składowa x_i była mała, to korygowana wartość w_i w niewielkim tylko stopniu przyczyniła się do powstania usuwanego błędu δ. W szczególności, jeśli $x_i = 0$, to odpowiadająca składowa wektora wcale nie będzie zmieniana — zupełnie słusznie, gdyż nie uczestniczyła w obliczaniu błędnej wartości y. Warto także zauważyć, że w przypadku *ujemnych* wartości x_i odpowiednie składowe $\mathbf{W}$ będą *zmniejszane*, a nie powiększane (oczywiście przy założeniu, że $\delta > 0$). Oznacza to, że w odróżnieniu od wejść „pobudzających", które dla uzyskania większego sygnału y trzeba wzmocnić, wejścia „hamujące" trzeba właśnie osłabić.

Tego rodzaju impresje oparte na strukturze wzoru opisującego uczenie neuronu można by snuć jeszcze długo. Nie będziemy tu tego robili ze względu na brak miejsca, zachęca się jednak Czytelnika, by poświęcił nieco czasu i sam przemyślał różne kombinacje sygnałów wejściowych, wag i błędów, upewniając się, że w każdym przypadku efekty działania, opisanej wyżej reguły uczenia są możliwe do intuicyjnego uzasadnienia. Takie rozważania nie są oczywiście *dowodem*, że metoda jest poprawnie zbudowana (dowód taki skonstruujemy za chwilę), jednak znaczenia intuicji nie można nie doceniać. Żadna metoda matematyczna, nawet bardzo wszechstronnie podbudowana dowodami, nie znajdzie uznania w technice, jeśli nie będzie zgodna z intuicją i zdrowym rozsądkiem. Cenimy bowiem bardzo matematykę, jako narzędzie i pomocniczy instrument inżyniera, jednak wyżej cenimy własny intelekt,

który nie powinien się buntować przeciwko wnioskom formulowanym z wykorzystaniem matematyki. Inżynier bowiem zawsze odpowiada za jakość zbudowanej przez siebie konstrukcji i w razie katastrofy to on jest stawiany przed sądem, a nie jakieś tam równanie!

3.4 Matematyczne aspekty procesu uczenia sieci

Skupmy teraz uwagę na matematycznych właściwościach opisanego procesu uczenia. Przytoczony wyżej wzór opisujący regułę uczenia elementu ADALINE jest bardzo ważny, gdyż zastosowana w nim reguła DELTA jest podstawą i punktem wyjścia przy konstrukcji większości algorytmów automatycznego uczenia. Z tego względu warto dokonać także matematycznej analizy istoty tej reguły. Dla skupienia uwagi podczas dalszych rozważań wygodnie jest wprowadzić pojęcie *ciągu uczącego*. Ciąg ten ma następującą budowę:

$$U = << \mathbf{X}^{(1)}, z^{(1)} >, < \mathbf{X}^{(2)}, z^{(2)} >, \ldots, < \mathbf{X}^{(N)}, z^{(N)} >>$$

czyli składa się z par postaci $< \mathbf{X}^{(j)}, z^{(j)} >$ zawierających wektor $\mathbf{X}$ podany w j-tym kroku procesu uczenia i informację o wymaganej odpowiedzi neuronu z w tym kroku. Uwzględniając numerację par składających się na ciąg uczący, można zapisać w zmodyfikowany sposób rozważaną tu regułę uczenia:

$$\mathbf{W}^{(j+1)} = \mathbf{W}^{(j)} + \eta^{(j)}\, \delta^{(j)}\, \mathbf{X}^{(j)}$$

We wzorze tym

$$\delta^{(j)} = z^{(j)} - y^{(j)}$$

gdzie

$$y^{(j)} = \mathbf{W}^{(j)} * \mathbf{X}^{(j)}$$

Reguła ta daje się łatwo stosować pod warunkiem wprowadzenia początkowego wektora wag $\mathbf{W}^{(1)}$; zwykle zakłada się, że wektor ten ma składowe wybrane losowo[2]. W programach symulujących pracę sieci neuronowych wykorzystuje się specjalne rozkazy, nakazujące nadanie losowych wartości składowym wektora wag, ale nie jest to konieczne, ponieważ mogą one pochodzić na przykład z poprzedniego cyklu uczenia, kiedy neuronowi narzucano realizację innej funkcji. Jedyne, czego trzeba bezwarunkowo unikać, to przyjmowania jednakowych wartości dla różnych składowych wektora $\mathbf{W}$ na początku procesu uczenia (powinno się zapewnić $\forall_{\mu,\nu}\ w_\mu^{(1)} \neq w_\nu^{(1)}$). Nie dotrzymanie tego warunku prowadzi niekiedy do braku postępów w początkowym etapie procesu uczenia.

Po wprowadzeniu przytoczonych oznaczeń można sformułować *cel* procesu uczenia. Celem tym jest uzyskanie zgodności odpowiedzi neuronu $y^{(j)}$ z wymaganymi wartościami $z^{(j)}$, co daje się sprowadzić do minimalizacji funkcji kryterialnej[3]

$$Q = \frac{1}{2} \sum_{j=1}^{N} \left(z^{(j)} - y^{(j)} \right)^2$$

[2] Dodatkowo celowe jest przyjmowanie na początku niezbyt dużych wartości składowych wektora W, tak, aby norma $\| W^{(1)} \|$ była ograniczona. Wymaganie to wynika z faktu, że stosowanie reguły DELTA w procesie uczenia prowadzi do systematycznego zwiększania normy wektora W ($\| W^{(j+1)} \| > \| W^{(j)} \|$) co może prowadzić do przepełnienia (*overflow*) podczas prowadzenia stosownych obliczeń. Przyjęcie małej wartości $\| W^{(1)} \|$ wprawdzie nie eliminuje tego problemu, ale może spowodować, że wystąpi on znacznie później.

[3] Funkcja Q nawiązuje do szeroko znanej metody najmniejszych kwadratów (oznaczanej w literaturze jako LMS), dlatego omawiana metoda uczenia bywa także opisywana w skrócie jako metoda LMS.

Wzór ten wygodnie będzie zapisać w postaci

$$Q = \sum_{j=1}^{N} Q^{(j)}$$

gdzie oczywiście

$$Q^{(j)} = \frac{1}{2}\left(z^{(j)} - y^{(j)}\right)^2$$

Ponieważ $Q = Q(\mathbf{W})$, zatem poszukiwanie minimum może być dokonywane metodą gradientową. Skupiając uwagę na i-tej składowej wektora $\mathbf{W}$, możemy więc zapisać:

$$w_i' - w_i = \Delta w_i = -\eta \frac{\partial Q}{\partial w_i}$$

Wzór ten można interpretować w sposób następujący: poprawka Δw_i jakiej powinna podlegać i-ta składowa wektora $\mathbf{W}$, musi być proporcjonalna do i-tej składowej gradientu funkcji Q. Znak - w omawianym wzorze wynika z faktu, że gradient Q wskazuje kierunek najszybszego **wzrastania** tej funkcji, podczas gdy w omawianej metodzie zależy nam na tym, by zmieniać $\mathbf{W}$ w kierunku najszybszego **malenia** błędu popełnianego przez sieć, czyli w kierunku najszybszego **malenia** funkcji Q. Współczynnik proporcjonalności η określa wielkość kroku Δw i może być w zasadzie wybierany dowolnie. W rzeczywistości jednak podlega on pewnym ograniczeniom. Przyjmując, że uczenie jest procesem stochastycznym (co znajduje uzasadnienie w stochastycznym charakterze ciągu uczącego U), możemy stwierdzić, że wartości η powinny być zależne od numeru pokazu j, a zatem powinny tworzyć ciąg $< \eta^{(1)}, \eta^{(2)}, \eta^{(3)}, \ldots, >$. Uwzględniono to we wzorze wprowadzającym regułę uczenia, pisząc $\eta^{(j)}$ zamiast η. Z teorii aproksymacji stochastycznej można wyprowadzić wniosek, że $\eta^{(j)}$ powinny spełniać warunki

$$\sum_{j=1}^{\infty} \eta^{(j)} = \infty$$

oraz

$$\sum_{j=1}^{\infty} \left(\eta^{(j)}\right)^2 < \infty$$

W najprostszym wypadku warunki te spełnia ciąg

$$\eta^{(j)} = \frac{\eta^{(0)}}{j}$$

ale szybkie malenie $\eta^{(j)}$ ze wzrostem j jest w tym wariancie czynnikiem zbyt silnie ograniczającym efektywny czas uczenia (już dla niewielkich numerów j wartości $\eta^{(j)}$ stają się mniejsze od dokładności numerycznej używanego komputera i w kolejnych pokazach proces uczenia praktycznie zatrzymuje się, gdyż $\eta^{(j)} = 0$). Z tego względu, a także w celu ograniczenia[4] wzrostu modułu wektora $\mathbf{W}$, proponuje się niekiedy w charakterze ciągu $\eta^{(j)}$ wartości obliczane ze wzoru

$$\eta^{(j)} = \left(\sum_{i=1}^{j} \|\mathbf{X}^{(i)}\|^2\right)^{-1}$$

[4] Ograniczenie wzrostu modułu W jest celowe, ponieważ nieograniczenie rosnące bezwzględne wartości składowych w_i prowadzą do trudności z ich praktyczną realizacją zarówno przy symulacji komputerowej (problem dokładności numerycznej) jak i przy fizycznym odwzorowaniu wag w elektronicznym modelu neuronu.

albo prościej

$$\eta^{(j)} = \frac{\lambda}{\|\mathbf{X}^{(j)}\|^2}$$

gdzie λ jest pewną ustaloną stałą (zwykle $0,1 < \lambda < 1$). W praktyce jednak najczęściej rezygnuje się z tych subtelności i arbitralnie przyjmuje się pewną ustaloną wartość η, niezależną od numeru kroku j. Pamiętać tylko trzeba, że wybór zbyt wielkiego kroku Δw grozi tym, że „przeskoczymy" właściwe rozwiązanie i proces uczenia nie będzie zbieżny, natomiast zbyt małe η prowadzi do bardzo wolnego (nieefektywnego) procesu uczenia. Zwyczajowo przyjmuje się $\eta = 0,6$ i obserwując proces uczenia sieci, ustala się, czy wartośc ta może być utrzymana przez czas dłuższy, czy też powinna być zmniejszona.

Powróćmy jednak do analizy procesu uczenia jako gradientowej minimalizacji funkcji kryterialnej (funkcji błędu). Rozpisując wzór gradientowego uczenia dla kroku j, otrzymujemy:

$$w_i^{(j+1)} - w_i^{(j)} = \Delta w_i^{(j)} = -\eta \frac{\partial Q^{(j)}}{\partial w_i}$$

Uwzględniając fakt, że Q zależne jest od y, a dopiero y jest funkcją wektora wag $\mathbf{W}$, możemy zapisać wzór, odpowiadający pochodnej funkcji złożonej:

$$\frac{\partial Q^{(j)}}{\partial w_i} = \frac{\partial Q^{(j)}}{\partial y^{(j)}} \frac{\partial y^{(j)}}{\partial w_i}$$

Na podstawie zależności $Q^{(j)} = \frac{1}{2}(z^{(j)} - y^{(j)})^2$ natychmiast można ustalić, że

$$\frac{\partial Q^{(j)}}{\partial y^{(j)}} = -(z^{(j)} - y^{(j)}) = -\delta^{(j)}$$

natomiast liniowa funkcja wiążąca sygnal wyjściowy $y^{(j)}$ z wektorem $wag\mathbf{W}^{(j)}$ powoduje, że oczywście

$$\frac{\partial y^{(j)}}{\partial w_i} = x_i^{(j)}$$

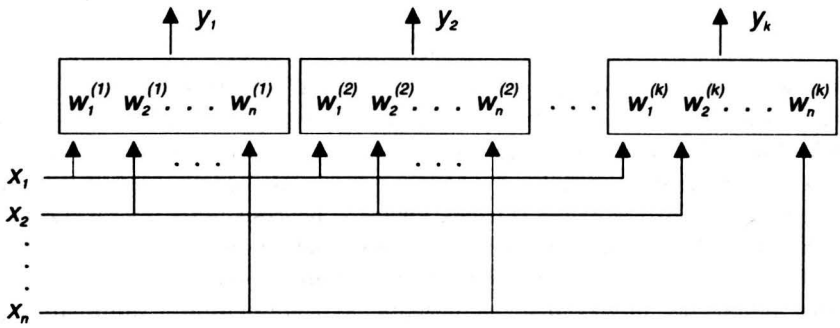
Zbierając wszystkie przedstawione wyniki razem, uzyskuje się potrzebną formule uczenia neuronu typu ADALINE:

$$\Delta w_i^{(j)} = \eta\, \delta^{(j)}\, x_i^{(j)}$$

co potwierdza jej poprawność. Ponieważ funkcja jakości Q jest unimodalna (jest to paraboloida eliptyczna z jednym tylko dobrze określonym minimum) — proces uczenia jest zbieżny i pozwala wyznaczyć potrzebny wektor wag $\mathbf{W}^*$, zapewniający dokladną realizację przez neuron wymaganej funkcji $y = f(\mathbf{X})$, jeśli jest to funkcja liniowa lub gwarantujący optymalną (w sensie minimum błędu średniokwadratowego) aproksymację tej funkcji, jeśli jest ona nieliniowa.

3.5 Uczenie sieci elementów liniowych

W sposób analogiczny do wyżej opisanego algorytmu uczenia pojedynczego neuronu można uczyć także całą sieć elementów liniowych.



Obiektem podlegającym uczeniu jest w tym wypadku *macierz* $\mathbf{W}_k$, a ciąg uczący ma postać

$$U = << \mathbf{X}^{(1)}, \mathbf{Z}^{(1)} >, < \mathbf{X}^{(2)}, \mathbf{Z}^{(2)} >, \ldots, < \mathbf{X}^{(N)}, \mathbf{Z}^{(N)} >>$$

gdzie $\mathbf{Z}^{(j)}$ są k-elementowymi wektorami oznaczającymi wymagane zestawy odpowiedzi sieci na wymuszenia danych odpowiednimi wektorami $\mathbf{X}^{(j)}$. Sieć taka w literaturze nazywana jest MADALINE (*Many* ADALINE *s*). Uczenie sieci MADALINE odbywa się w sposób całkowicie analogiczny do wyżej opisanego, z tą tylko różnicą, że formula uczenia ma w tym wypadku postać macierzową:

$$\mathbf{W}_k^{(j+1)} = \mathbf{W}_k^{(j)} + \eta \left(\mathbf{Z}^{(j)} - \mathbf{Y}^{(j)}\right) \left(\mathbf{X}^{(j)}\right)^T$$

Warto skupić przez chwilę uwagę na wymiarach macierzy i wektorów wchodzących w skład powyższego wzoru. Wektor $\mathbf{X}$ jest n elementowy i po transpozycji tworzy wiersz o n kolumnach. Wektory $\mathbf{Z}$ oraz $\mathbf{Y}$ są k elementowe, więc ich różnica tworzy kolumnę o k wierszach. Przemnożenie tych elementów przez siebie tworzy macierz poprawek $\Delta\mathbf{W}_k$ o rozmiarach $[k \times n]$ — dokładnie zgodną z wymiarami macierzy $\mathbf{W}_k$, co umożliwia poprawne dodawanie tych macierzy.

Przypominając interpretację funkcjonowania sieci o n wejściach i k wyjściach jako *filtru* przetwarzającego sygnały $\mathbf{X}$ na odpowiadające im (zgodnie z określonym odwzorowaniem) sygnały $\mathbf{Y}$ — można teraz rozważać proces uczenia sieci jako adaptację filtru do określonych potrzeb. Wkraczamy w ten sposób w obszerny krąg zagadnień *filtracji adaptacyjnej* oraz zbliżamy się do problemu poszukiwania *optymalnego filtru* (z punktu widzenia określonych kryteriów), samoczynnie dostosowującego się do potrzeb. Sieci neuronowe (zwłaszcza typu MADALINE) były i są chętnie stosowane jako filtry adaptacyjne (na przykład do eliminacji efektu „echa" w liniach telefonicznych) i jest to jeden z ważniejszych obszarów zastosowań neurokomputerów. Filtry tworzone w wyniku uczenia sieci neuronowej mogą być wykorzystywane do typowych zadań (dolno lub górnoprzepustowa filtracja sygnału, eliminacja zakłóceń, polepszanie stosunku sygnału do szumu, wydobywanie określonych cech sygnału, analiza widmowa itp.), ale mogą też mieć zupełnie nowe zastosowania.

Filtry tego typu mogą służyć na przykład do odtwarzania kompletnego sygnału na podstawie jego fragmentu (na przykład odnalezienie w banku danych portetu, w sytuacji kiedy

pokazano jedynie oczy i usta). Takie zadanie nazywa się generalnie „pamięcią adresowaną kontekstowo" lub pamięcią asocjacyjną (w odróżnieniu od typowej pamięci komputera, w której dostęp do określonych informacji waunkowany jest znajomością przypisanego im adresu lub klucza. Pamięć asocjacyjna to jeden z najbardziej obiecujących kierunków rozwoju sieci neuronowych. Pamięć taka pozwala odtworzyć cały zestaw zapamiętanych informacji w przypadku przedstawienia informacji niekompletnej lub niedokładnej. Tego rodzaju urządzenie, działając w powiązaniu z *systemem ekspertowym*, może doskonale odpowiadać na pytania użytkowników. Zadaniem sieci jest w tym wypadku uzupełnienie wiadomości podanych przez użytkownika w taki sposób, by system ekspertowy mógł (w oparciu o bazę wiedzy) efektywnie szukać rozwiązania.

Innym zastosowaniem omawianych tu filtrów może być tak zwany „filtr nowości" (na przykład do wyświetlania na ekranie jedynie tych fragmentów sceny, które uległy nagłej zmianie, co znajduje zastosowanie w nowoczesnej ochronie obiektów przed włamaniem). Filtr tego typu może mieć także szerokie zastosowanie w automatyce przemysłowej. Do zagadnienia tego powrócimy dalej.

3.6 Uczenie bez nauczyciela

Opisany wyżej schemat uczenia sieci MADALINE opierał się silnie na założeniu, że istnieje zewnętrzny arbiter („nauczyciel"), który podaje poprawne odpwiedzi $\mathbf{Z}$, w wyniku czego korekta macierzy wag $\mathbf{W}_k$ postępuje w sposób sterowany i zdeterminowany przez cel, jakim jest minimalizacja błędu Q. Taki schemat (nazywany w literaturze *supervised learning* albo od struktury stosowanego algorytmu *delta rule*) jest czasem niewygodny, ponieważ wymaga udziału nauczyciela w procesie uczenia, a ponadto jest mało wiarygodny biologicznie, gdyż wielu czynności mózg uczy się bez świadomego i celowego instruktażu. Dlatego w literaturze przedmiotu, wiele uwagi poświęca się opracowanej przez Hebba [Hebb49] i formalnie dopracowanej przez Suttona [Sutt81] technice uczenia bez nauczyciela, zwanej *unsupervised learning* lub *hebbian learning*. Zasada tego uczenia polega na tym, że waga $w_i^{(m)}$, i-tego wejścia m-tego neuronu wzrasta podczas prezentacji j-tego wektora wejściowego $\mathbf{X}^{(j)}$ proporcjonalnie do iloczynu i-tej składowej sygnału wejściowego $x_i^{(j)}$ docierającego do rozważanej synapsy i sygnału wyjściowego $y_m^{(j)}$ rozważanego neuronu. Zapisując to w formie wzoru, otrzymujemy:

$$w_i^{(m)(j+1)} = w_i^{(m)(j)} + \eta\, x_i^{(j)} y_m^{(j)}$$

przy czym oczywiście

$$y_m^{(j)} = \sum_{i=1}^{n} w_i^{(m)(j)}\, x_i^{(j)}$$

Podwójne górne indeksy przy współczynnikach wagi $w_i^{(m)(j)}$ wynikają z faktu, że trzeba uwzględnić numerację neuronów, do których wagi te należą (m) oraz numerację kroków, wynikających z kolejnych pokazów.

Podobnie jak w przypadku opisywanej wyżej reguły DELTA spróbujmy znaleźć intuicyjną interpretację dla opisywanej reguły. Wzmocnieniu ulegają w niej te wagi, które są aktywne (duże $x_i^{(i)}$) w sytuacji, gdy „ich" neuron jest pobudzony (duże $y_m^{(j)}$). Tego typu sieć jest zatem *autoasocjacyjna*: jeśli pewien wzór pobudzeń $\mathbf{X}$ sygnalizowany jest przez pewne m-te wyjście sieci, to w miarę upływu czasu ta sygnalizacja staje się coraz bardziej wyraźna. Sieć

uczy się w ten sposób rozróżniać nadchodzące do niej bodźce i grupować je w pewne kategorie (*cluster*), gdyż neuron wytrenowany do rozpoznawania pewnego sygnału $\mathbf{X}^{(j)}$ będzie zdolny do rozpoznawania także tych sygnałów, które są **podobne** do tego wzorcowego. Ta zdolność do uogólniania zdobytego doświadczenia jest jedną z najważniejszych cech sieci neuronowej — oczywiście nie tylko przy omawianej tu regule uczenia. Regułę uczenia pochodzącą od praw Hebb nazywa się także niekiedy uczeniem korelacyjnym (*correlation learning*), ponieważ zmierza ona do takiego dopasowania wag, aby uzyskać najlepszą korelację między sygnałami wejściowymi, a zapamiętanym w formie wartości wag „wzorcem" sygnału, na który określony neuron ma reagować. Ta terminologia ma pewną zaletę, gdyż eksponuje fakt *uśredniania* wejściowych sygnałów przez sieć uczoną według reguły Hebba. Istotnie, jeśli w ciągu uczącym znajdzie się kilka egzemplarzy sygnałów $\mathbf{X}^{(j)}$ reprezentujących ten sam obiekt, lecz różniących się drobnymi szczegółami — wówczas sieć wytworzy pewien uogólniony wzorzec sygnału, a sygnał wyjściowy rozważanego neuronu y_m będzie miarą podobieństwa kolejnych sygnałów do tego uśrednionego wzorca. W wyniku tego uzyskuje się pewną generalizację doświadczeń zdobywanych przez sieć w trakcie procesu uczenia, dzięki czemu sieć zdolna jest także rozpoznawać obiekty, których jej nigdy wcześniej nie pokazywano (na przykład zniekształcone, niekompletne lub zakłócone przez szum).

Sieć ucząca się (bez udziału nauczyciela), według reguły Hebba, osiąga na ogół dobre wyniki i całkiem samoczynnie grupuje wejściowe sygnały w „kategorie" sensownie odpowiadające klasom podobnych do siebie sygnałów wejściowych $\mathbf{X}$. Efekt ten nie jest jednak nigdy *pewny*, gdyż dość istotnie zależy od początkowego stanu sieci (początkowych, przypadkowych wartości $w_i^{(m)(1)}$) decydującego o tym, jak w początkowym etapie uczenia zaczną krystalizować się ośrodki przyszłych grup. Niewielka jest szansa na to, by różne klasy podobieństwa istniejące wśród wejściowych sygnałów od razu odnalazły „swoje" neurony, zatem trzeba się liczyć z tym, że na tę samą klasę „wskazywać" będzie kilka neuronów. Oznacza to, że liczba neuronów w sieci k musi być większa, niż oczekiwana liczba rozróżnianych klas, a ponadto nie pozwala z góry przewidzieć, który neuron nauczy się sygnalizować którą klasę. Posługując się przykładem, na który często powołują się autorzy odpowiednich prac, można oczekiwać, że sieć trenowana w celu rozpoznawania ręcznie pisanych liter nauczy się je identyfikować, ale nie wiemy z góry, który neuron sygnalizować będzie **A**, a który **B**, a także nie mamy gwarancji, że każdej literze odpowiadać będzie tylko jeden neuron — bardziej prawdopodobne jest, że kilka neuronów nauczy się rozpoznawać jako „swoje" **A** oraz (co gorsza) może się zdarzyć, że żaden neuron nie nauczy się rozpoznawać **C**.

3.7 Warianty metod uczenia i samouczenia

Proces samouczenia i samoorganizacji można uczynić bardziej efektywnym poprzez zastosowanie tak zwanego przyrostowego samouczenia (*differential hebbian learning*), polegającego na uzależnieniu procesu zmiany wag od *przyrostów* sygnałów wejściowych na danej synapsie i wyjściowych, na danym neuronie:

$$w_i^{(m)(j+1)} = w_i^{(m)(j)} + \eta \left[(x_i^{(j)} - x_i^{(j-1)})(y_m^{(j)} - y_m^{(j-1)}) \right]$$

W niektórych wypadkach ta przyrostowa strategia daje znacznie lepsze rezultaty niż „czysty" algorytm Hebba. Inna modyfikacja algorytmu samouczenia pochodzi z wczesnych prac Grossberga [Gros74] i znana jest w literaturze jako koncepcja „gwiazdy wejść" (*instar training*). W koncepcji tej wybiera się pewien neuron (powiedzmy o numerze m) i narzuca się

mu taką strategię uczenia, by zapamiętał i potrafił potem rozpoznać aktualnie wprowadzany sygnał **X**. Inne neurony sieci są w tym czasie bezczynne. Omówiona zasada uczenia może być zapisana jako

$$w_i^{(m)(j+1)} = w_i^{(m)(j)} + \eta^{(j)} \left(x_i^{(j)} - w_i^{(m)(j)}\right)$$

Wydaje się, że po wszystkim co do tej pory napisano — nie ma już potrzeby komentowania tego prostego wzoru. Można tylko dodać, że z praktyki stosowania metody wynikają pewne zalecenia odnośnie wyboru wartości $\eta^{(j)}$, które powinny zmieniać się zgodnie z empiryczną regułą:

$$\eta^{(j)} = 0,1 - \lambda\, j$$

przy czym współczynnik λ trzeba wybrać na tyle mały, by w ciągu całego uczenia zachodził warunek $\eta^{(j)} > 0$.

Na zasadzie analogii do *instar* można wprowadzić także drugą propozycję Grossberga — koncepcję *outstar*. W koncepcji tej rozważa się wagi wszystkich neuronów całej warstwy, jednak wybiera się wyłącznie wagi łączące te neurony *z pewnym ustalonym wejściem*. W sieciach wielowarstwowych wejście to pochodzi od pewnego ustalonego neuronu wcześniejszej warstwy i to właśnie ten neuron staje się „gwiazdą wyjść" — *outstar*. Wzór opisujący uczenie sieci *outstar* jest następujący:

$$w_i^{(m)(j+1)} = w_i^{(m)(j)} + \eta^{(j)} \left(y_m^{(j)} - w_i^{(m)(j)}\right)$$

Warto raz jeszcze podkreślić odmienność powyższego wzoru w stosunku do poprzednio rozważanych. Tutaj i jest ustalone (arbitralnie lub w jakiś inny sposób), natomiast m jest zmienne i przebiega wszystkie możliwe wartości ($m = 1, 2, \ldots, k$). Reguła zmieniania $\eta^{(j)}$ jest w tym wypadku nieco inna, niż przy *instar* i może być dana wzorem:

$$\eta^{(j)} = 1 - \lambda\, j$$

O ile metoda *instar* stosowana jest w przypadku, kiedy trzeba sieć nauczyć rozpoznawania określonego sygnału **X**, o tyle metoda *outstar* znajduje zastosowanie przy uczeniu sieci wytwarzania określonego wzorca zachowań **Y** w odpowiedzi na określony sygnał inicjujący x_i.

Reguły uczenia podane przez Hebba i Grossberga były przez wielu badaczy wzbogacane i modyfikowane. Pierwsza interesująca modyfikacja polega na wprowadzeniu dyskryminacji wejściowych i wyjściowych sygnałów. Dyskryminacja ta polega na dzieleniu sygnałów $x_i^{(j)}$ oraz $y_m^{(j)}$ na „aktywne" i „nie aktywne". W tym celu używa się parametru $\varepsilon^{(j)}$ będącego ustaloną lub zależną od j wartością progową i wprowadza się nowe zmienne $\hat{x}_i^{(j)}$ oraz $\hat{y}_m^{(j)}$ zdefiniowane w sposób następujący:

$$\hat{x}_i^{(j)} = \begin{cases} 1 & \text{gdy } x_i^{(j)} > \varepsilon \\ 0 & \text{w przeciwnym przypadku} \end{cases}$$

oraz

$$\hat{y}_m^{(j)} = \begin{cases} 1 & \text{gdy } y_m^{(j)} > \varepsilon \\ 0 & \text{w przeciwnym przypadku} \end{cases}$$

Przy tych oznaczeniach nową technikę uczenia wg zmodyfikowanego algortmu Hebba zapisać można w postaci:

$$w_i^{(m)(j+1)} = w_i^{(m)(j)} + \eta\, \hat{x}_i^{(j)} \hat{y}_m^{(j)}$$

Warto zauważyć, że przy takim postawieniu sprawy współczynnik wagowy może być zmieniony wyłącznie o watość η (zawsze taką samą), przy czym o tym, czy zmiana jest dokonywana, decyduje w istocie pewien *logiczny* warunek, wyrażający się stwierdzeniem, aktywnemu sygalowi wejściowemu na danej synapsie $\hat{x}_i^{(j)}$ odpowiada aktywny sygnal wyjściowy $\hat{y}_m^{(j)}$.

Podany wyżej wzór pozwala wyłącznie na *zwiększanie* wartości wag, co może prowadzić do osiągania przez nie bardzo dużych wartości. Z tego względu proponuje się niekiedy uogólniony algorytm, w którym przy aktywnym wyjściu $\hat{y}_m^{(j)}$ synapsy obslugujące aktywne wejścia $\hat{x}_i^{(j)}$ uzyskują wzmocnienie, a synapsy obslugujące nieaktywne wyjścia są (w tym samym stopniu) oslabiane. Odpowiednia formula uczenia może być zapisana w formie:

$$w_i^{(m)(j+1)} = w_i^{(m)(j)} + \eta\, x_i^{(j)}\,(2\hat{y}_m^{(j)} - 1)$$

Warto zauważyć, że funkcjonowanie tego wzoru w istotny sposób oparto na fakcie, że $\hat{y}_m^{(j)} \in \{0, 1\}$. Omawiany wzór nazywany bywa regulą *Hebb/Anti-Hebb*. Jeszcze dalsze poszerzenie rozważanej metody uczenia daje wzór Hopfielda, w podobny sposób traktujący zarówno $\hat{x}_i^{(j)}$ jak i $\hat{y}_m^{(j)}$:

$$w_i^{(m)(j+1)} = w_i^{(m)(j)} + \eta\,(2\hat{x}_i^{(j)} - 1)(2\hat{y}_m^{(j)} - 1)$$

Intencja stosowania tego wzoru jest oczywista: wszystkie współczynniki wagowe podlegają przy nim bardzo intensywnemu treningowi, niezależnie od tego, czy dotyczą wejść aktywnych i niezależnie od tego, czy odpowiednie wyjścia są aktywne, natomiast oczywiście kierunek zmian wartości wag zależy od aktywności wejść i wyjść — co latwo można prześledzić w podanym wzorze zestawiając wszystkie cztery możliwe kombinacje wartości $\hat{x}_i^{(j)}$ i $\hat{y}_m^{(j)}$.

3.8 Uczenie z rywalizacją i sieci Kohonena

Istnieje jeszcze jedno interesujące podejście do zagadnienia samouczenia sieci typu MADALINE — tak zwane uczenie z rywalizacją (*competitive learnig*). Wprowadzil je Kohonen [Koho75] przy tworzeniu sieci neuronowych uczących się realizacji dowolnych odwzorowań $\mathbf{X} \Rightarrow \mathbf{Y}$. Z punktu widzenia tu prowadzonych rozważań można przyjąć, że zasada tego uczenia sieci Kohonena jest identyczna z regulą *instar*

$$w_i^{(m^*)(j+1)} = w_i^{(m^*)(j)} + \eta^{(j)}\,(\tilde{x}_i^{(j)} - w_i^{(m^*)(j)})$$

z dwoma dość istotnymi uzupelnieniami. Po pierwsze wektor wejściowy $\mathbf{X}$ jest przed procesem uczenia normalizowany tak, aby jego długość $\|\mathbf{X}\| = 1$. Oznacza to, że

$$\tilde{x}_i^{(j)} = \frac{x_i^{(j)}}{\sqrt{\sum_{\nu=1}^{n} (x_\nu^{(j)})^2}}$$

Po drugie, numer poddawanego treningowi neronu neuronu m^* nie jest już przypadkowy czy arbitralnie wybierany, jest to bowiem ten (i tylko ten) neuron, którego sygnal wyjściowy $y_{m^*}^{(j)}$ jest największy. Oznacza to, że przy każdorazowym podaniu sygnalu wejściowego $\mathbf{X}^{(j)}$ neurony rywalizują ze sobą i wygrywa ten, który uzyskal największy sygnal wyjściowy $y_{m^*}^{(j)}$. Tylko ten „zwycięski" neuron podlega następnie uczeniu, którego efektem jest jeszcze lepsze

dopasowanie wektora wag $\mathbf{W}^{(m^*)(j+1)}$ do rozpoznawania obiektów podobnych do $\mathbf{X}^{(j)}$. Inne neurony podlegać będą uczeniu przy pokazie — być może — innych obiektów $\mathbf{X}^{(\nu)}$, w tym kroku uczenia ich wektory wag pozostają nie zmienione.

Regula uczenia Kohonena bywa często wzbogacana o dodatkowy element związany z topologią uczącej się sieci. Neurony w sieci są **uporządkowane** (czego wyrazem są nadawane im numery m). Można więc wprowadzić pojęcie **sąsiedztwa** — na przyklad uznając za sąsiednie te neurony, które mają numery m, różniące się o ustaloną małą wartość (np. o jeden). Uogólniona regula samoorganizacji sieci Kohonena polega na tym, że uczeniu podlega nie tylko neuron m^* „wygrywający" w konkurencji z innymi neuronami sieci, ale także neurony, które z nim sąsiadują. Formalnie regulę tę można zapisać wzorem

$$w_i^{(m)(j+1)} = w_i^{(m)(j)} + \eta^{(j)}\, h(m, m^*)(\tilde{x}_i^{(j)} - w_i^{(m)(j)})$$

formula uczenia może być zapisana w formie:

$$w_i^{(m)(j+1)} = w_i^{(m)(j)} + \eta\, \hat{x}_i^{(j)}\, (2\hat{y}_m^{(j)} - 1)$$

Warto zauważyć, że funkcjonowanie tego wzoru w istotny sposób oparto na fakcie, że $\hat{y}_m^{(j)} \in \{0, 1\}$. Omawiany wzór nazywany bywa regulą *Hebb/Anti-Hebb*. Jeszcze dalsze poszerzenie rozważanej metoformie dyskretnej, na przyklad

$$h(m, m^*) = \begin{cases} 1 & \text{dla} \quad m = m^* \\ 0,5 & \text{dla} \quad |m - m^*| = 1 \\ 0 & \text{dla} \quad |m - m^*| > 1 \end{cases}$$

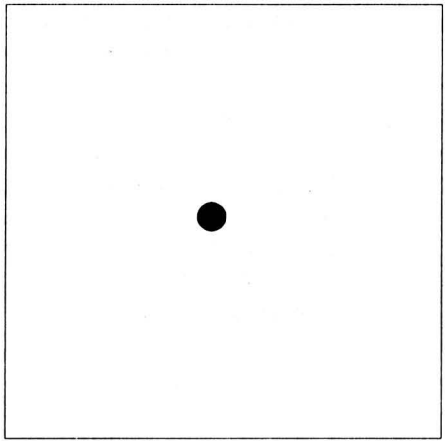
natomiast w bardziej skomplikowanych zadaniach funkcja $h(m, m^*)$ może być wyrażona za pomocą odległości $\rho(m, m^*)$, na przyklad

$$h(m, m^*) = \frac{1}{\rho(m, m^*)}$$

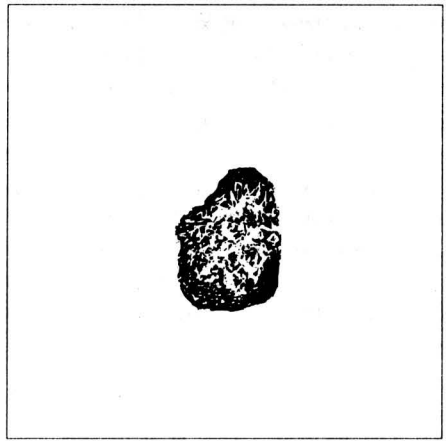
albo

$$h(m, m^*) = \exp\,(-[\rho(m, m^*)]^2)$$

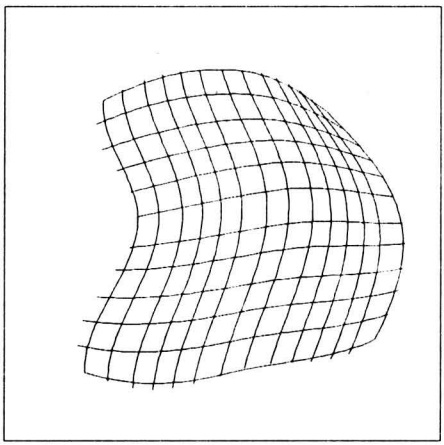
Oczywiście możliwe są także znacznie bardziej wyrafinowane funkcje, obejmujące szerszy zakres numerów m, a także uwzględniające róxne możliwe formy pojęcia sąsiedztwa. Najciekawsze efekty uzyskuje się wprowadzając metody Kohonena dla sieci neuronowych dwuwymiarowych, to znaczy takich, w których neurony ulożone są w strukturę tablicy o pewnej liczbie wyróżnionych wierszy i kolumn. Dla tego typu sieci, zbudowanych wedlug zasady kompetycyjnego uczenia z uwzględnieniem **dwuwymiarowego** sąsiedztwa, opisywane są w literaturze bardzo interesujące formy zachowania. Sieci tego typu są w stanie samorzutnie tak się organizować, by odwzorować strukturę wejściowego dwuwymiarowego obiektu — na przyklad samorzutnie utworzyć siatkę równomiernie pokrywającą określony kształt analizowanego obszaru. Podane niżej rysunki ilustrują ten proces dla jednego z przykladowych zadań.



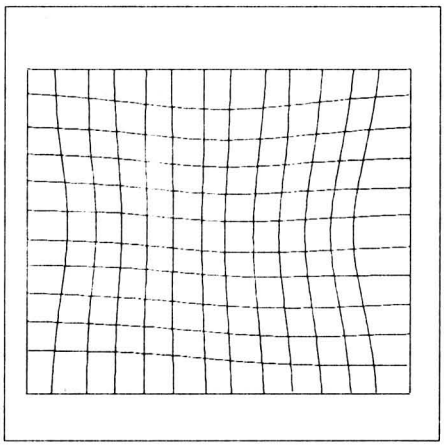
0



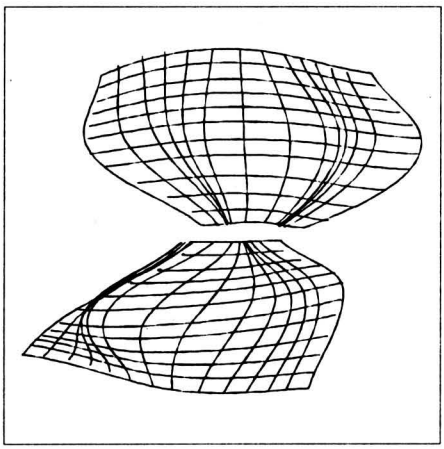
20



100



100 000



20 000

3.9 Uczenie z forsowaniem

Opisane wyżej techniki samouczenia (czy też „uczenia bez nauczyciela") mają bardzo interesującą odmianę, pozwalającą na wykorzystanie przytoczonych metod także w przypadku kiedy wektor wymaganych wartości sygnałów wyjściowych sieci $\mathbf{Z}^{(j)}$ jest znany (czyli w omawianym na samym początku zadaniu uczenia na podstawie ciągu uczącego U). Wszystkie omówione wyżej metody uczenia dadzą się wtedy bardzo efektywnie zastosować przy zastosowaniu prostej zamiany odpowiednich y przez stosowne z. Takie uczenie ma charakter „forsowania" poprawnych rozwiązań bez względu na to, co robi sieć. Metoda taka może mieć niekiedy istotną przewagę nad metodą DELTA, ponieważ nie wymaga w sposób jawny wyliczania wartości błędów popełnianych przez neurony sieci, co może bardzo istotnie przyspieszyć proces uczenia. Ponieważ istnieje głęboka i łatwa do prześledzenia analogia omawianych metod „forsownego uczenia z nauczycielem" do dyskutowanych wyżej metod „uczenia bez nauczyciela"— ograniczymy się jedynie do przytoczenia odpowiednich wzorów, pomijając tym razem dyskusję i próby interpretacji tych wzorów. Wyróżnić przy tym można następujące metody:

- metoda autoasocjacji:

$$w_i^{(m)(j+1)} = w_i^{(m)(j)} + \eta\, x_i^{(j)} z_m^{(j)}$$

- metoda przyrostowej autoasocjacji:

$$w_i^{(m)(j+1)} = w_i^{(m)(j)} + \eta \left[(x_i^{(j)} - x_i^{(j-1)})(z_m^{(j)} - z_m^{(j-1)}) \right]$$

- metoda zbliżania wektora wag do wektora odpowiedzi:

$$w_i^{(m)(j+1)} = w_i^{(m)(j)} + \eta^{(j)} \left(z_m^{(j)} - w_i^{(m)(j)} \right)$$

Wprowadzając podobnie jak wyżej oznaczenie

$$\hat{z}_m^{(j)} = \begin{cases} 1 & \text{gdy } y_m^{(j)} > \varepsilon \\ 0 & \text{w przeciwnym przypadku} \end{cases}$$

oraz korzystając z wprowadzonej wyżej definicji $\hat{x}_i^{(j)}$, możemy podać kolejne trzy dalsze reguły uczenia:

$$\begin{aligned} w_i^{(m)(j+1)} &= w_i^{(m)(j)} + \eta\, \hat{x}_i^{(j)}\, \hat{z}_m^{(j)} \\ w_i^{(m)(j+1)} &= w_i^{(m)(j)} + \eta\, \hat{x}_i^{(j)}\, (2\hat{z}_m^{(j)} - 1) \\ w_i^{(m)(j+1)} &= w_i^{(m)(j)} + \eta\, (2\hat{x}_i^{(j)} - 1)(2\hat{z}_m^{(j)} - 1) \end{aligned}$$

Wybór jednej z podanych wyżej możliwości podyktowany być musi każdorazowo oceną ich przydatności w konkretnym zadaniu, przy czym z powodu braku ogólnej teorii konieczne są eksperymenty i poszukiwania oparte na badaniach empirycznych.

Rola „forsownego uczenia" może być dokładniej prześledzona dla dwóch interesujących teoretycznie i przydatnych praktycznie konkretnych sytuacji. Przepiszmy równanie autoasocjacji z podanej wyżej formy skalarnej

$$w_i^{(m)(j+1)} = w_i^{(m)(j)} + \eta\, x_i^{(j)} z_m^{(j)}$$

do postaci **macierzowej**:

$$\mathbf{W}_k^{(j+1)} = \mathbf{W}_k^{(j)} + \eta\, \mathbf{Z}^{(j)} \left[\mathbf{X}^{(j)}\right]^T$$

Efekt uczenia można wówczas zapisać w postaci sumarycznego wzoru

$$\mathbf{W}_k = \sum_{j=1}^{N} \eta\, \mathbf{Z}^{(j)} \left[\mathbf{X}^{(j)}\right]^T + \mathbf{W}_k^{(1)}$$

Załóżmy, że $\mathbf{W}_k^{(1)} = 0$ oraz przyjmijmy, że wszystkie wektory wejściowe w ciągu uczącym są **ortonormalne**, to znaczy

$$\forall_j \left[\mathbf{X}^{(j)}\right]^T \mathbf{X}^{(m)} = \begin{cases} 1 & \text{gdy } j = m \\ 0 & \text{gdy } j \neq m \end{cases}$$

Wówczas sieć nauczy się wiernie odtwarzać wymagane sygnały wyjściowe dla wszystkich rozważanych sygnałów wejściowych. Łatwo się o tym przekonać: załóżmy, że macierz $\mathbf{W}_k$ jest już „nauczona" zgodnie z podanym wyżej wzorem i załóżmy, że w trakcie „egzaminu" pojawił się wektor sygnałów wejściowych $\mathbf{X}$, identyczny z jednym z wcześniej przedstawionych wektorów uczących $\mathbf{X}^{(m)}$. Obliczając sygnał wyjściowy „nauczonej" sieci otrzymujemy:

$$\boldsymbol{Y} = \boldsymbol{W}_k\, \boldsymbol{X} = \sum_{j=1}^{N} \eta\, \boldsymbol{Z}^{(j)} \left[\boldsymbol{X}^{(j)}\right]^T \boldsymbol{X} = \sum_{j=1}^{N} \eta \boldsymbol{Z}^{(j)} \left[\boldsymbol{X}^{(j)}\right]^T \boldsymbol{X}^{(m)} =$$

$$= \eta\, \boldsymbol{Z}^{(m)} \left[\boldsymbol{X}^{(m)}\right]^T \boldsymbol{X}^{(m)} = \eta\, \boldsymbol{Z}^{(m)}$$

Jak widać, sieć jest w tym wypadku zdolna dokładnie odtworzyć zapamiętany sygnał, zatem może służyć jako pamięć. Mało tego, sieć jest także zdolna do uogólniania sygnałów wejściowych. Załóżmy, że zbiór uczący jest tak zbudowany, że wejściowe sygnały $\mathbf{X}^{(j)}$ mogą być traktowane jako przypadkowo zakłócone realizacje pewnego idealnego wzorca $\mathbf{X}$

$$\mathbf{X}^{(j)} = \mathbf{X} + \xi^{(j)}$$

gdzie składnik $\xi^{(j)}$ reprezentuje „szum" zniekształcający wejściowe sygnały w każdym kolejnym prezentowanym przykładzie. Taki model jest dość wiarygodny w wielu konkretnych zadaniach, na przykład przy rozpoznawaniu ręcznie pisanych liter, wektor $\mathbf{X}^{(j)}$ reprezentujący kształt kolejnej prezentowanej litery **A**, może być rozważany jako suma wektora cech $\mathbf{X}$ idealnej (takiej z podręcznika kaligrafii) litery **A** oraz przypadkowych zniekształceń pochodzących od indywidualnych cech pisma osoby piszącej $\xi^{(j)}$. Jeśli dla każdego z rozważanych wektorów wejściowych $\mathbf{X}^{(j)}$ podawać będziemy ten sam wektor wyjściowy $\mathbf{Z}$ (ponieważ wszystkie te próbki prezentują w istocie przypadkowe odmiany tego samego obrazu), wówczas w wyniku procesu uczenia macierz wag zostanie zbudowana w następujący sposób:

$$\boldsymbol{W}_k = \sum_{j=1}^{N} \eta \boldsymbol{Z} \left[\boldsymbol{X}^{(j)}\right]^T = \sum_{j=1}^{N} \eta \boldsymbol{Z} \left[\boldsymbol{X} + \boldsymbol{\xi}^{(j)}\right]^T =$$

$$= \eta \boldsymbol{Z} \left(N \boldsymbol{X}^T + \sum_{j=1}^{N} [\boldsymbol{\xi}^{(j)}]^T\right)$$

Jak widać w macierzy wag manifestować się będzie głównie idealny wzorzec $\mathbf{X}$, ponieważ jego wartość mnożona jest przez $\mathbf{N}$, którego wartość w ogólnym wypadku może być bardzo duża, podczas gdy suma

$$\sum_{j=1}^{N} \left[\xi^{(j)}\right]^T$$

ma na ogół niewielką wartość, ponieważ poszczególne składniki $\xi^{(j)}$ jako wartości przypadkowe mogą się wzajemnie kompensować. Oznacza to, że sieć ma zdolność uśredniania wejściowych sygnałów i może samodzielnie „odkryć" powtarzalny wzorzec $\mathbf{X}$ w serii przypadkowo zniekształconych wejściowych obserwacji $\mathbf{X}^{(j)}$.

Oba omawiane wyżej efekty są niezależne od siebie nawzajem i mogą być „nałożone", co w wyniku daje sieć zapamiętującą różne reakcje $\mathbf{Z}$ na różne sygnały wejściowe $\mathbf{X}$ wydobywane spośród serii przypadkowo zniekształconych obserwacji. Jedynym ograniczeniem jest przy tym „pojemność pamięci", związana z liczbą neuronów sieci. Przyjmuje się, że dla sieci liczącej k neuronów maksymalna liczba możliwych do zapamiętania wzorców wyraża się przybliżonym empirycznym wzorem

$$\mathbf{N}_{max} \cong \frac{k}{2 \log k}$$

3.10 Przyspieszanie procesu uczenia

W bardziej złożonych i rozbudowanych sieciach istotną rolę odgrywa sprawność procesu uczenia. Trzeba z naciskiem stwierdzić, że większość omówionych wyżej metod uczenia daje pozytywny końcowy wynik dopiero po prezentacji bardzo dużej (często rzędu setek tysięcy!) liczby prezentowanych pokazów. Dlatego w literaturze poświęconej sieciom neuronowym opisuje się liczne metody zwiększania szybkości procesu uczenia. Jedną z metod jest odpowiedni dobór wartości $\eta^{(j)}$, o czym była już wyżej mowa. Drugą, powszechnie stosowaną „sztuczką" jest wprowadzanie do wzoru na korektę wektora wag dodatkowego składnika, uwzględniającego „bezwładność" procesu zmiany wag w postaci tak zwanego *momentum* (termin ten jest opowiednikiem polskiego pojęcia fizycznego „pęd", ale operowanie „pędem" w wielu kontekstach związanych z uczeniem sieci jest nieco niewygodne i dlatego w tej książce termin *momentum* będzie używany w oryginalnym brzmieniu angielskim). Używając zapisu wektorowego i odwołując się do najprostszej metody uczenia z nauczycielem za pomocą prostej reguły DELTA składnik momentum można wprowadzić w następujący sposób:

$$\boldsymbol{W}_k^{(j+1)} = \boldsymbol{W}_k^{(j)} + \eta_1 \left(\boldsymbol{Z}^{(j)} - \boldsymbol{Y}^{(j)}\right) \left[\boldsymbol{X}^{(j)}\right]^T + \eta_2 \boldsymbol{M}^{(j)}$$

gdzie momentum $\mathbf{M}^{(j)}$ wyliczane jest ze wzoru

$$\mathbf{M}^{(j)} = \mathbf{W}_k^{(j)} - \mathbf{W}_k^{(j-1)}$$

Z doświadczeń autora książki oraz z obszernych danych literaturowych wynika, że wprowadzenie składnika momentum wpływa na zwiększenie szybkości uczenia oraz zdecydowanie polepsza stabilność tego procesu, czego bezpośrednim wyrazem może być możliwość bezpiecznego zwiększenia wartości współczynnika η. Z doświadczeń wynika, że bardzo dobre wyniki procesu uczenia można uzyskać przyjmując $\eta_1 = 0,9$ i $\eta_2 = 0,6$. Czasami stosuje się także i w tym algorytmie zmniejszanie wartości η w kolejnych krokach j, ale z reguły zachowuje się przy tym stałą wartość stosunku η_1/η_2.

Inną techniką zwiększającą szybkość procesu uczenia jest ograniczanie go wyłącznie do **dużych** poprawek. Oznacza to, że reguła uczenia ma dodatkowy parametr η_3 i działa według następującej zasady (podanej niżej przy pominięciu — dla skrócenia zapisu — składnika

momentum):

$$w_i^{(m)(j+1)} = \begin{cases} w_i^{(m)(j)} + \eta_1 \, x_i^{(j)} \, (z_m^{(j)} - y_m^{(j)}) & \text{gdy} \quad (z_m^{(j)} - y_m^{(j)}) > \eta_3 \\ w_i^{(m)(j)} & \text{gdy} \quad (z_m^{(j)} - y_m^{(j)}) \leq \eta_3 \end{cases}$$

Z opisywanych w literaturze doświadczeń wynika, że parametr η_3 powinien na początku procesu uczenia przyjmować wartość 0,2, a następnie powinien być redukowany do zera [Klima91].

Wybór współczynników η_1, η_2 i η_3 jest w chwili obecnej w większym stopniu sprawą sztuki (a więc na przykład intuicji eksperymentatora), niż sprawą ścisłej wiedzy. Zwykle należy zaczynać od dużych wartości tych współczynników i potem stopniowo je redukować, czasami jednak proces uczenia lepiej przebiega, gdy w pierwszych krokach współczynniki te mają bardzo małe wartości, dopiero potem zwiększane są do typowego (wyżej podanego) poziomu. Jest to kwestia doświadczenia i wyczucia, gdyż trudno znaleźć teoretyczne uzasadnienie dla tego zjawiska. Podobnie powszechnie akceptowana empirycznie sprawdzona reguła, głosząca, że wymienione współczynniki powinny mieć mniejsze wartości dla dużych sieci (w sensie liczby neuronów), nie może być chwilowo przekonywująco wyjaśniona na podstawie czystej teorii. Inna reguła, mająca duże znaczenie dla sieci złożonych z **nieliniowych** neuronów, głosi konieczność stosowania mniejszych wartości tych współczynników dla warstw sieci bliższych wejścia systemu i większych dla warstw dalszych — także nie całkiem wiadomo dlaczego, chociaż sam fakt nie podlega dyskusji.

Sejnowski i Rosneberg zaproponowali w 1987 roku jeszcze inną metodę przyspieszania procesu uczenia i polepszania jego zbieżności. Metoda ta, zwana czasem **wygładzaniem wykładniczym**, opiera się na wzorze podobnym w zasadzie do wzoru zawierającego *momentum*, ale zakładającego nieco inne „ważenie" składników pochodzących od poprzedniej korekty wektora wagi i od określonego w kroku j błędu popełnianego przez sieć:

$$\boldsymbol{W}_k^{(j+1)} = \boldsymbol{W}_k^{(j)} + \eta_1 \left[(1 - \eta_2) \, (\boldsymbol{Z}^{(j)} - \boldsymbol{Y}^{(j)}) \, [\boldsymbol{X}^{(j)}]^T + \right.$$
$$\left. + \eta_2 \, (\boldsymbol{W}_k^{(j)} - \boldsymbol{W}_k^{(j-1)}) \right]$$

Poważnym problemem przy uczeniu sieci neuronowych (nie tylko typu MADALINE) jest zapewnienie odpowiedniej randomizacji zbioru uczącego U. Chodzi o to, że kolejne elementy $< \mathbf{X}^{(j)}, \mathbf{Z}^{(j)} >$ powinny być *losowo* wybierane, a tymczasem z reguły ciąg uczący prezentowany jest *cyklicznie*, ponieważ liczba obiektów zarejestrowanych w tym ciągu (oznaczana N) jest zbyt mała w porównaniu z liczbą pokazów niezbędnych dla prawidłowego uczenia sieci. Można wprawdzie przed każdym cyklem mieszać starannie pokazy wchodzące w skład ciągu uczącego, ale nie zawsze daje to wystarczająco dobre rezultaty. W takim wypadku polepszenie efektów procesu uczenia można osiągnąć za pomocą *kumulowania błędów*. Technika ta polega na podzieleniu przedziału zmienności numerów obiektów ciągu uczącego j na odcinki o założonej długości η_4 (z praktyki wynika, że najlepsze wyniki daje $30 \leq \eta_4 \leq 50$), ponumerowanie tych odcinków pomocniczymi indeksami $j^*(j^* = 0, 1, 2, \ldots)$ i obliczanie w każdym z tych odcinków pomocniczej wartości $\mathbf{S}^{(j^*)}$

$$\boldsymbol{S}^{(j^*)} = \sum_{j=\eta_4 j^*+1}^{\eta_4(j^*+1)} \eta_1 \left(\boldsymbol{Z}^{(j)} - \boldsymbol{Y}^{(j)} \right) \left[\boldsymbol{X}^{(j)} \right]^T$$

nazywanej skumulowanym błędem. Wartości skumulowanego błędu wykorzystywane są do obliczania nowych wartości macierzy wag

$$\mathbf{W}_k^{(j^*+1)} = \mathbf{W}_k^{(j^*)} + \mathbf{S}^{(j^*)}$$

przy czym poprawki te są wnoszone jedynie w momentach j spełaniających równanie

$$j = \eta_4 j^* \quad , \quad j^* = 1, 2, \ldots$$

a więc znacznie rzadziej, niż w przypadku prowadzenia uczenia co krok (tzn. dla każdego j).

3.11 Uwagi końcowe

Sieci MADALINE (wraz ze swymi odmianami) były pierwszymi efektywnie zastosowanymi sieciami neuronowymi i mimo ogromnego postępu notowanego w tej dziedzinie, pozostają wciąż bardzo użytecznym narzędziem — szczególnie przy budowie systemów rozpoznających, filtrów adaptacyjnych, pamięci asocjacyjnych itp. Jednak **możliwości sieci budowanych z elementów liniowych są ograniczone**, jak to słusznie podkreślili **Minsky i Papert** w swojej znanej książce [Mins69].

Ograniczenie to ma dwojakiego rodzaju charakter. Po pierwsze odwzorowania $\mathbf{X} \Rightarrow \mathbf{Y}$ jakie może realizować sieć MADALINE są wyłącznie odwzorowaniami liniowymi. Wynika to w trywialny sposób z zasady działania tej sieci. Po drugie, klasa dostępnych odwzorowań nie zależy od tego, czy mamy do czynienia z siecią jedno- czy wielowarstwową. Uprzedzając nieco wiadomości wprowadzone w następnych rozdziałach stwierdzić należy, że sieci neuronowe *wielowarstwowe* mają w ogólnym przypadku znacznie bogatsze możliwości, niż sieć jednowarstwowa. Nie dotyczy to jednak niestety sieci typu MADALINE. Rozważmy bowiem przykładowo sieć dwuwarstwową. Niech sygnał wejściowy do pierwszej warstwy sieci dany będzie (jak dotychczas) wektorem $\mathbf{X}$ o n elementach. Jeśli pierwsza warstwa ma k neuronów a komplet wag dla tej warstwy wyraża się macierzą $\mathbf{W}_k$, wówczas sygnał wyjściowy z tej warstwy daje się zapisać wektorem $\mathbf{Y}$ o k składowych, który można wyznaczyć formalnie z używanej wyżej zależności

$$\boldsymbol{Y} = \boldsymbol{W}_k \, \boldsymbol{X}$$

Dotychczas na tym się nasza analiza kończyła. Załóżmy jednak, że sieć ma kolejną warstwę zawierającą m neuronów, do których doprowadzone są sygnały składające się na wektor $\mathbf{Y}$. Macierz współczynników wagowych dla tej warstwy sieci oznaczmy przez $\mathbf{W}_m$, ma ona oczywiście wymiary $[m \times k]$. W następstwie działania tej warstwy sygnał $\mathbf{Y}$ o k składowych przetworzony zostaje w sygnał $\mathbf{U}$ o m składowych zgodnie ze wzorem

$$\boldsymbol{U} = \boldsymbol{W}_m \, \boldsymbol{Y}$$

Jednak drogą prostego podstawienia można uzyskać formułę bezpośrednio wiążącą wejście całej dwuwymiarowej sieci $\mathbf{X}$ z jej wyjściem $\mathbf{U}$:

$$\boldsymbol{U} = \boldsymbol{W}_k \, \boldsymbol{W}_m \, \boldsymbol{X}$$

Jednak macierze $\mathbf{W}_k$ i $\mathbf{W}_m$ można wymnożyć (warto sprawdzić, że jest to dozwolone ze względu na zgodne z wymogami rachunku macierzowego dopasowanie wymiarów tych macierzy), w wyniku czego można zapisać odwzorowanie w postaci

$$\boldsymbol{U} = \boldsymbol{W}_{km} \, \boldsymbol{X}$$

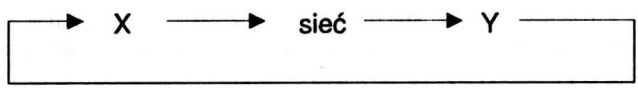
gdzie $\mathbf{W}_{km}$ jest iloczynem macierzy $\mathbf{W}_k$ i $\mathbf{W}_m$. Oznacza to, że związek pomiędzy wektorami **X** i **U** zadany jest w formie liniowego odwzorowania niczym istotnym nie różniącego się od tego występującego przy sieci jednowarstwowej. Tak więc przy sieciach MADALINE budowa wielowarstwowych struktur jest bezcelowa, ponieważ sieć o wielu warstwach ma dokładnie takie same możliwości, jak sieć jednowarstwowa.

Na podstawie podanego wyżej stwierdzenia, że sieć neuronowa (liniowa!) wielowarstwowa jest funkcjonalnie równoważna sieci jednowarstwowej, a także na podstawie konstatacji oczywistego faktu, że sieć tego typu realizuje jedynie odwzorowania **liniowe**, sformułowano w przeszłości kategoryczny i — na szczęście — niesłuszny pogląd o ograniczonej przydatności sieci neuronowych jako systemów przetwarzania informacji [Mins69]. Stwierdzenia tego bowiem absolutnie nie można stosować do sieci nieliniowych, opisanych w następnych rozdziałach.

Pozostając jeszcze przy temacie sieci liniowych, opisywanych równaniem macierzowym

$$\boldsymbol{Y} = \boldsymbol{W}_n \boldsymbol{X}$$

gdzie wektory **X** i **Y** są n-wymiarowe, a macierz $\mathbf{W}_n$ ma rozmiary $[n \times x]$, możliwe jest rozpatrywanie sieci dwojakiego typu. Prostsze w analizie i częściej stosowane są sieci z jednokierunkowym przepływem sygnałów *(feedforward)*, tzn. takie, w których sygnały **X** znajdujące się na wejściu, są niezależne od sygnałów wyjściowych **Y** pojawiających się na wyjściu. Sieci takie nazywane są czasami heteroasocjacyjnymi *(heteroassociative)* i z założenia rozpatrywane są jako twory statyczne, gdyż ewentualne procesy przejściowe, zachodzące w sieci podczas jej pracy, nie mają znaczenia z punktu widzenia celu funkcjonowania sieci i mogą być pomijane. Natomiast w niektórych zastosowaniach istotną rolę odgrywają sieci ze sprzężeniami zwrotnymi *(feedback)*, w których sygnały wyjściowe **Y** są pośrednio lub bezpośrednio podawane na wejście **X**. Takie sieci określane bywają jako autoasocjacyjne *(autoassociative)* i odznaczają się bogatymi własnościami dynamicznymi, ponieważ sygnały w pętli



mogą krążyć dowolnie długo powodując powstawanie różnorodnych i ciekawych przebiegów przejściowych, których absolutnie pomijać nie wolno, ponieważ stanowią one istotę działania sieci. Dokładniej o sieciach tego typu mowa będzie w rozdziale dotyczącym tzw. sieci Hopfielda, które są najbardziej znanym i najważniejszym przykładem sieci autoasocjacyjnych.

Rozdział 4

Nieliniowe sieci neuronowe

4.1 Nieliniowy model neuronu

Siecią neuronową, która odegrala historycznie bardzo istotną rolę byl niewątpliwie PERCEPTRON Rosenblatta [Rose62][1]. Ta klasyczna konstrukcja (z 1957 roku) zawierala bardzo wiele oryginalnych i ciekawych elementów (na przyklad losowy mechanizm doboru połączeń między neuronami), których jednak nie będziemy tu szczegółowo dyskutowali, gdyż w rozwoju neurodynamiki stanowily jedynie epizod. Skupimy natomiast uwagę na najbardziej istotnym nowym elemencie, wnoszonym przez PERCEPTRON w stosunku do omówionej wyżej ADALINE i MADALINE: nieliniowym elemencie przetwarzającym informację w każdym neuronie sieci. Wprowadzenie do sieci neuronowej nieliniowego elementu przetwarzającego informacje jest uzasadnione, ponieważ rzeczywiste biologiczne neurony są nieliniowe. Nieliniowy element przyjmowany w sieciach neuronowych może być opisany równaniem

$$y = \varphi(e)$$

gdzie $\varphi(.)$ jest wybraną funkcją nieliniową a sygnal e odpowiada łącznemu pobudzeniu neuronu, zgodnemu z formulą przyjmowaną uprzednio dla ADALINE

$$e = \sum_{i=1}^{n} w_i \, x_i$$

lub uzupelnioną dodatkowo o skladnik staly *(bias)*

$$e = \sum_{i=1}^{n} w_i \, x_i + w_0$$

W dalszych rozważaniach będziemy zawsze zakladali, że skladnik w_0 występuje — najwyżej ma wartość 0. Aby jednak uprościć wszystkie występujące dalej wzory i wyprowadzenia,

[1] Jako ciekawostkę warto odnotować fakt, że pierwowzorem perceptronu byla ucząca się maszyna *(learning machine)* zbudowana w 1951 roku przez studentów uniwersytetu Princeton, **Marvina Minskiego** i **Deana Edmondsa**. Ciekawostka polega na tym, że Minski byl potem „grabarzem" idei perceptronu, którą totalnie skrytykował w swojej sławnej książce [Mins69].

przyjmiemy, że obok sygnałów $< x_1, x_2, \ldots, x_n >$ składających się na wektor $\mathbf{X}$, występować będzie składnik $x_0 \equiv 1$. Ten formalny zabieg pozwoli zapisać sygnał e prostym wzorem

$$e = \sum_{i=0}^{n} w_i \, x_i$$

w którym na uwagę zasługuje zakres sumowania zaczynający się od zera. Będziemy nadal stosowali zapis wektorowy używając wcześniej wprowadzonych oznaczeń $\mathbf{W}$ i $\mathbf{X}$,

$$e = \mathbf{W}^T \mathbf{X}$$

teraz jednak stale będziemy uważali, że są to wektory o $n + 1$ elementach, zawierające składowe x_0 oraz w_0.

Podana wyżej postać formuły określającej sygnał wypadkowego pobudzenia neuronu nie jest jedyną możliwą. Używane bywają bowiem także i inne formuły, na przykład sumy kumulowanej, której wartość w j-tym kroku symulacji może być wyznaczona ze wzoru

$$e^{(j+1)} = e^{(j)} + \sum_{i=1}^{n} w_i^{(j)} \, x_i^{(j)}$$

funkcji majoryzacji

$$e = \sum_{i=1}^{n} \mu_i,$$

gdzie μ_i jest miarą efektywności i-tego wejścia wyznaczaną ze wzoru

$$\mu_i = \begin{cases} 1 & \text{gdy} \quad w_i \, x_i > 0 \\ 0 & \text{gdy} \quad w_i \, x_i \leq 0 \end{cases}$$

maksimum

$$e = MAX_i \; w_i \, x_i$$

lub minimum

$$e = MIN_i \; w_i \, x_i,$$

a w niektórych zastosowaniach przydatna jest produktowa (oparta na iloczynie) miara łącznego pobudzenia:

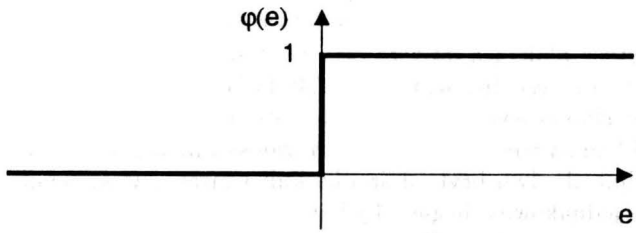
$$e = \prod_{i=1}^{n} w_i \, x_i$$

Te i inne funkcje scalające wejściowe sygnały x_i w łączne wypadkowe pobudzenie e, używane są w perceptronie jedynie jako wstępny etap przetwarzania informacji w neuronie.

O specyficznych właściwościach perceptronu decyduje funkcja φ określająca nieliniowy związek między sygnałem wypadkowego pobudzenia neuronu e, a jego odpowiedzią y. W najbogatszej reprezentowanej anglojęzycznej literaturze przyjęło się nazywać sygnał e *NET value*, a funkcję φ — *activation function*. W klasycznym perceptronie funkcja φ ma postać progową:

$$\varphi(e) = \begin{cases} 1 & \text{gdy} \quad e \geq 0 \\ 0 & \text{gdy} \quad e < 0 \end{cases}$$

pokazaną na rysunku:



Taka postać ma sporo wad, od niej jednak zaczniemy dyskusję, gdyż przy tej postaci funkcji najlatwiej jest wprowadzić pewne intuicje przydatne potem przy dyskusji wszelkich nieliniowych sieci neuronowych. Zacznijmy od odnotowania najbardziej naturalnego faktu: ponieważ sygnal wyjściowy perceptronu przyjmuje dyskretne wartości ($y = 1$ lub $y = 0$), przeto może być rozważany w kategoriach określonej *decyzji* (na przyklad — obiekt należy do rozpoznawanej klasy lub obiekt do niej nie należy). Możliwa jest także interpretacja oparta na logice matematycznej — sygnal $y = 1$ można wówczas interpretować jako „prawda", a sygnal $y = 0$ jako „falsz". Przy pierwszej interpretacji perceptron może być traktowany jako uklad dokonujący pewnego podzialu wejściowego zbioru sygnalów $\mathbf{X}$ na dwie klasy: klasę wyróżnioną (dla której $y = 1$) oraz klasę odrzuconą. Przy drugiej interpretacji perceptron może być rozpatrywany jako uklad realizujący pewną funkcję logiczną, a więc automat skończony. Druga interpretacja jest wcześniejsza, w istocie pierwsze prace dotyczące sieci neuronowych **McCullocha i Pittsa** [McCu43] taką wlaśnie proponowaly interpretację, a rozwijana na gruncie logiki matematycznej teoria funkcji progowych stanowi dla tego podejścia wygodny fundament matematyczny. Obszerniejsze informacje na ten temat zawarto w książce [Tade91a].

4.2 Właściwości nieliniowego modelu neuronu

Z dyskretnymi wartościami sygnalu wyjściowego z nieliniowego modelu neuronu wiąże się następujące zagadnienie. Zależnie od postaci przyjętej funkcji $\varphi(e)$ sygnal y można rozpatrywać jako **binarny** $y \in \{0, 1\}$ lub **bipolarny** $y \in \{-1, 1\}$. Na pozór jest to różnica malo istotna, gdyż za pomocą trywialnego przeskalowania można oczywiście swobodnie przejśc od jednej postaci sygnalu do drugiej. Jednak w rzeczywistości różnica może być dość istotna, ponieważ punkty należące do zbioru $\{0, 1\}$ są wierzcholkami jednostkowego hipersześcianu w $\mathcal{R}^n$, natomiast punkty należące do zbioru $\{-1, 1\}$ leżą na powierzchni jednostkowej sfery w $\mathcal{R}^n$. Intuicja wywodząca się z dwu- i trójwymiarowej przestrzeni podpowiada, że sfera jest podobna do sześcianu, oto sfera może być porównana z sześcianem o zaokrąglonych wierzcholkach. Tymczasem w n-wymiarowej przestrzeni sfera i sześcian różnią się w sposób zasadniczy, o czym można się przekonać porównując na przyklad objętość kuli o promieniu r z objętością sześcianu o boku l. Objętość sześcianu wynosi oczywiście $V_s = l^n$, natomiast objętość kuli wyraża się bardziej skomplikowaną formula:

$$V_k = \begin{cases} \frac{\pi^{n/2}}{(n/2)!} r^n & \text{gdy } n \text{ jest parzyste} \\ \frac{2^n \pi^{(n-1)/2}([n-1]/2)!}{n!} r^n & \text{gdy } n \text{ jest nieparzyste} \end{cases}$$

Rozważając sześcian o jednostkowej długości boku ($l = 1$) stwierdzamy oczywiście, że w każdej przestrzeni ma on jednostkową objętość ($V_s = 1$), podczas gdy kula o promieniu $r = 1$ ma objętość malejącą do zera ze wzrostem $n \to \infty$. Jeśli więc rozważymy w n-wymiarowej przestrzeni sześcian z wpisaną weń kulą, to w miarę wzrostu wymiaru przestrzeni n kula będzie coraz slabiej wypełniać sześcian, zatem podobieństwo tych dwóch bryl będzie coraz bardziej problematyczne. Do podobnego wniosku możemy dojść rozważając odległości punktów powierzchni obydwu bryl od środka kuli wpisanej w sześcian. Wszystkie punkty sfery są oczywiście jednakowo odlegle od jej środka (odległością jest promień sfery r), natomiast narożniki sześcianu (których liczba w n-wymiarowej przestrzeni wynosi 2^n) są odlegle od środka sześcianu o $\sqrt{n/2}$, czyli ze wzrostem wymiaru przestrzeni n oddalają się coraz bardziej od środka sześcianu. Ponieważ równocześnie objętość sześcianu pozostaje stała, zatem n-wymiarowy hipersześcian przypominać musi jeża o coraz większej liczbie coraz bardziej smukłych „kolców" (wierzchołków) i o coraz mniejszym „korpusie" w centralnej części. Odbiega to bardzo od potocznych obiegowych wyobrażeń, lecz w przestrzeniach o dużej liczbie wymiarów ituicja nie jest najlepszym doradcą, a pewnych „oczywistych" stwierdzeń nie da się mechanicznie przenosić, gdyż można latwo popełnić błąd. Na przykład płaszczyzny, których położenie w trójwymiarowej przestrzeni zwykle latwo sobie wyborazić już w przestrzeni czterowywmiarowej zachowują się osobliwie. Nie tylko mogą one być równolegle (tzn. nie mieć wspólnych punktów) lub przecinać się (tzn. mieć wspólną całą prostą, będącą krawędzią przecięcia), ale dodatkowo mogą mieć ze sobą dokladnie jeden wspólny punkt — co wymyka się wyobraźni, lecz jest bezspornym faktem[2]. Podobnie tzw. koła wielkie wyznaczone na sferze (takie jak południki na globusie), które w przestrzeni trójwymiarowej muszą się zawsze przecinać (np. na biegunach) w przestrzeni czterowymiarowej mogą być równolegle[3].

Wspomniane wyżej paradoksy trzeba mieć stale w pamięci usilując angażować wyobraźnię w opis zjawisk zachodzących w n-wymiarowych przestrzeniach, z którymi mamy stale do czynienia w analizie i opisie sieci neuronowych. Dopóki elementy sieci są liniowe — intuicja może być użyteczna, gdyż zjawiska opisywane np. na płaszczyźnie mają swoją w miarę naturalną kontynuację w przestrzeni o większej liczbie wymiarów. Z chwilą wprowadzenia funkcji $\varphi(e)$ jako funkcji nieliniowej — trzeba podchodzić do intuicyjnych wywodów z daleko posuniętą ostrożnością. Niżej przedstawione rozważania będą uwzględniały to zalecenie.

4.3 Właściwości nieliniowych sieci wielowarstwowych

Przyjmując interpretację progowej funkcji $\varphi(e)$ jako funkcji rozdzielającej przestrzeń wejściowych sygnalow $\mathbf{X}$ na obszar wyróżniony, w którym $y = 1$, oraz na resztę — stwierdzić należy, że przy przyjęciu najczęściej rozważanej reguły scalania wejściowych sygnałów w postaci

$$e = \sum_{i=0}^{n} w_i \, x_i$$

[2] Na przykład płaszczyzna $x_1 = x_2 = 0$ i płaszczyzna $x_3 = x_4 = 0$ mają dokladnie jeden wspólny punkt $(x_1 = x_2 = x_3 = x_4 = 0)$.

[3] Na przykład są równolegle (gdyż nie mają wspólnego punktu) koła wielkie dane równaniami $x_1 + x_2 = 1$ i $x_3 + x_4 = 1$.

podział ten formuje granica mająca postać *hiperpłaszczyzny*. Istotnie, jeśli

$$\varphi(e) = \begin{cases} 1 & \text{gdy } e \geq 0 \\ 0 & \text{gdy } e < 0 \end{cases}$$

to obszar, w którym neuron podejmuje decyzję $y = 1$ ogranicza powierzchnia $e = 0$, czyli twór o równaniu

$$\sum_{i=0}^{n} w_i x_i = 0$$

Jak latwo się przekonać dla $n = 2$ jest to równanie linii prostej, dla $n = 3$ — równanie plaszczyzny, a dla $n > 3$ twór nazywany prawidlowo *rozmaitością liniową stopnia* $n - 1$, a popularnie traktowany jako plaszczyzna w n-wymiarowej przestrzeni czyli w skrócie *hiperpłaszczyzna*.

Interpretacja dzialania neuronu budującego perceptron jako *dyskryminatora liniowego* pozwala zorientować się, jakie są możliwe formy odwzorowań wejściowego zbioru sygnalów $\mathbf{X} \in \mathbf{X}$ na dyskretny zbiór decyzji $\{0, 1\}$. Jak widać może on realizować wszystkie te odwzorowania, w których wystarczy wydzielenie podobszaru przestrzeni $\mathbf{X}$ mającego formę otwartej półprzestrzni odgraniczonej hieperplaszyzną. W malo interesującym teoretycznie, ale wygodnym do graficznej prezentacji przypadku $n = 2$ mamy do czynienia z półplaszczyzną odgraniczoną za pomocą linii prostej.

Latwo zauważyć, że proces uczenia, polegający zawsze na zmianie wartości współczynników w_i, pozwala wprawdzie ustawić graniczną hiperplaszczyznę w dowolnym położeniu, nie pozwala jednak na zmianę charakteru realizowanego odwzorowania, co powoduje, że pewnych typów odwzorowań nie da się uzyskać za pomocą neuronu typu perceptronowego bez względu na to, jak dlugo i jak wyrafinowanie by się go uczylo. Klasycznym przykładem tego typu nierozwiązywalnego zadania jest „problem XOR" wprowadzony przez **Minsky'ego** [Mins69]: Perceptron nie może nauczyć się realizacji odwzorowania

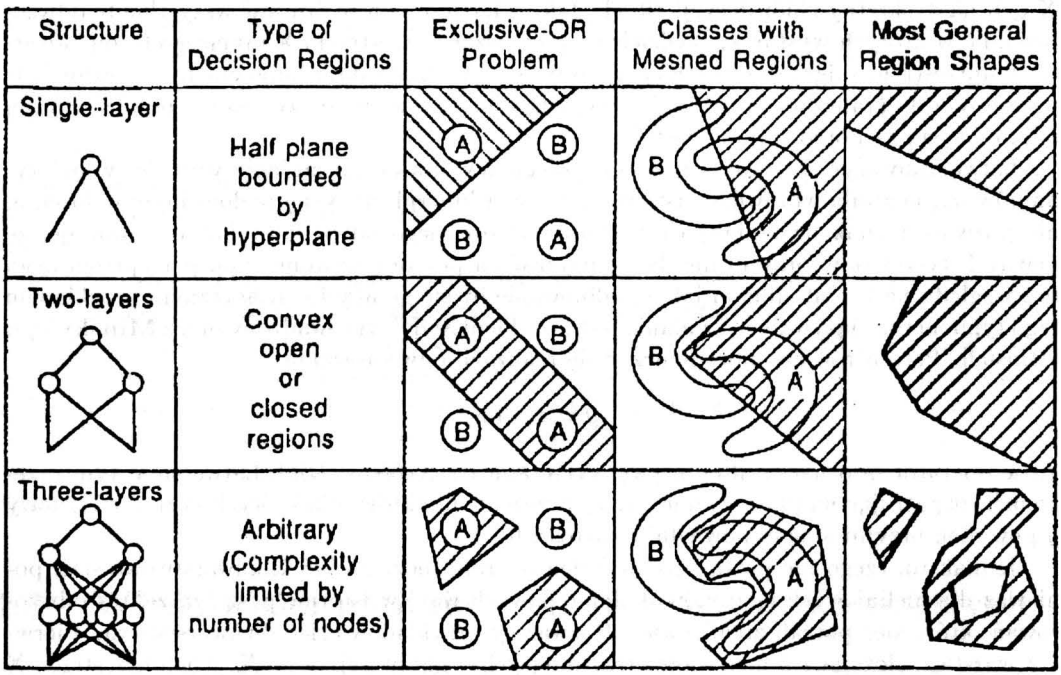
$$y = x_1 \oplus x_2$$

gdzie operator $\oplus$ oznacza alterantywę wyłączającą *(eXclusive OR)*. Latwo się o tym przekonać, rozpatrując na plaszczyźnie x_1, x_2 położenie punktów, dla których sygnal wyjściowy y powinien przyjmować odpowiednio wartości 0 i 1.

Jednak to, czego nie potrafi zrobić jeden neuron, może zrobić kilkuwarstwowa sieć, ponieważ dla nieliniowych neuronów dodanie nowych warstw istotnie poszerza zakres odwzorowań, które sieć potrafi zrealizować. Rozważmy przykladowo sieć dwuwarstwową. Pierwsza warstwa, złożona z k neuronów otrzymujących sygnaly wejściowe $\mathbf{X}$, dzieli przestrzeń $\mathbf{X}$ tych sygnalów za pomocą k oddzielnych hiperplaszczyzn. Powstaje w ten sposób uklad $2k$ liniowo rozdzielnych obszarów, które sygnalizowane są przez odpowednie zestawy 0 i 1 jako wartości sygnalów neuronów pierwszej warstwy. Sygnaly te podawane są z kolei na wejścia neuronów drugiej warstwy, które dokonują klasyfikacji zestawów tych sygnalów wedlug zasady: sygnal wyjściowy neuronu drugiej warstwy ma wartośc 0 lub 1 w zależności od tego, jaki **podzbiór** neuronów pierwszej warstwy sygnalizuje 0, a jaki 1. W efekcie neurony drugiej warstwy mogą rozpoznawać (sygnalizować) pojawienie się wektorów wejściowych $\mathbf{X}$ zawartych w pewnych ograniczonych obszarach przestrzenie $\mathbf{X}$. Obszary te nie muszą już być równoważne do calej półprzestrzeni $\mathbf{X}$, ponieważ możliwe jest sygnalizowanie bardziej złożonego podobszaru, ograniczonego z wielu stron fragmentami wielu hiperplaszczyzn.

Niestety, sieć dwuwarstwowa nie pozwala jeszcze na rozpoznawanie dowolnego podobszaru przestrzeni **X**, ponieważ łatwo sprawdzić, że obszary sygnalizowane przez neurony drugiej warstwy muszą być **wypukłe** oraz **jednospójne**. Takie wypukle, jednospójne, ograniczone hiperplaszczyznami (a więc „kanciate") obszary nazywa się zwykle **simpleksami** (od angielskiego *simplex* — prosty, nie skomplikowany), zatem sieć dwuwarstwowa rozpoznawać może tylko simpleksy. Jest to dość istotne ograniczenie, jednak łatwo sobie wyobrazić, co należy zrobić, żeby się od tego ograniczenia uwolnić. Oczywiście — trzeba użyć kolejnej warstwy neuronów. Ta trzecia warstwa otrzymuje na wejściu informacje o przynależności sygnałów **X** do wykrywanych przez drugą warstwę simpleksów i dokonuje na nich kolejnego etapu przetwarzania, łącząc je ze sobą lub odejmując je od siebie. W rezultacie możliwe jest utworzenie **dowolnych** obszarów: także niewypuklych (poprzez „wycinanie" jednego simpleksu innymi simpleksami) oraz niejednospójnych (gdy sumuje się na jednym neuronie trzeciej warstwy sygnały pochodzące od kilku rozłącznych simpleksów).

Możliwości te ilustruje zbiorczo poniższy rysunek:



Można więc powiedzieć, że za pomocą nieliniowej sieci neuronowej o przynajmniej trzech warstwach można zrealizować **dowolne odwzorowanie**, wiążące w całkowicie dowolny sposób wejściowe sygnały **X** z wyjściowymi sygnałami sieci. Jest to ważne stwierdzenie, gdyż ta „omnipotencjalność" nieliniowych sieci neuronowych (co najmniej trójwarstwowych) leży u podstaw ich szerokiego stosowania. Dletego poddamy te sieci bliższej analizie, rozważając kolejno wszystkie jej elementy.

4.4 Formy nieliniowości neuronu

Funkcja wiążąca łączne pobudzenie neuronu e z jego sygnalem wyjściowym y

$$y = \varphi(e)$$

opisana wyżej jako funkcja progowa, miewa także liczne modyfikacje, z których na szczególną uwagę zasluguje **sigmoidalna** funkcja wywodząca się z funkcji **logistycznej**

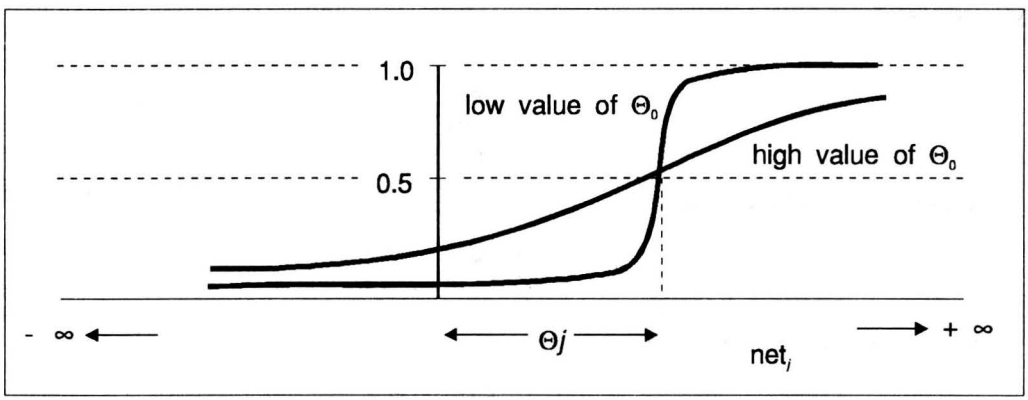
$$y = \frac{1}{1 + \exp(-\beta e)}$$

Zaletą tej funkcji jest prosta i latwa do obliczenia wartość jej pochodnej. Latwo wykazać, że w przypadku funkcji logistycznej

$$\frac{d\varphi}{de} = y(1 - y)$$

z czego będziemy dalej w sposób istotny korzystali.

Na marginesie tej dyskusji warto odnotować fakt, że w literaturze spotyka się różne oznaczenia funkcji $\varphi(e)$ i jej parametrów. Ilustruje to rysunek pokazujący przebieg funkcji logistycznej dla dwóch wartości parametru β, zaczerpnięty z książki [Hech90], na którym — zgodnie z tradycją literatury anglojęzycznej — sumaryczne pobudzenie e oznaczono jako net_j, β oznaczono jako θ_0 a parametr θ_j odpowiada w notacji przyjętej w tej książce wartości wyrazu wolnego w_0.



Ważną właściwością funkcji logistycznej jest fakt, że zbiór jej wartości należy do **otwartego** zbioru $y \in (0, 1)$, co oznacza, że wartości 0 i 1, mające istotne znaczenie przy niektórych interpretacjach funkcjonowania sieci nauronowych (np. teoria tzw. sieci logiki ciąglej, por. [Tade91a]) są tu nieosiągalne. Z tego względu mimo niewątpliwych zalet funkcji sigmoidalnej (logistycznej) rozważane są także inne funkcje opisujące nieliniową zależność między sumarycznym pobudzeniem, a sygnalem wyjściowym neuronu. Przykladowo, chętnie stosowaną funkcją jest **tangens hiperboliczny**

$$y = \tanh(\beta e)$$

który można rozpisać jako

$$y = \frac{\exp(\beta e) - \exp(-\beta e)}{\exp(\beta e) + \exp(-\beta e)}.$$

Przy zastosowaniu tej funkcji $y \in (-1, 1)$. Zaletą funkcji tangens hiperboliczny jest także prosta formula, określająca pochodną tej funkcji w zależności od jej wartości

$$\frac{d\varphi}{de} = (1 + y)(1 - y)$$

Ta formula, podobnie jak wcześniej przytoczony wzór dla funkcji logistycznej, bardzo ułatwia stosowanie odpowiednich funkcji w trakcie procesu uczenia.

Aby uzyskać wartości z przedziału domkniętego $y \in [-1, 1]$ stosuje się niekiedy funkcję **sinus**

$$y = \sin(\beta e).$$

a dokładniej fragment sinusoidy połączony z płaskimi asymptotami rozciągającymi dziedzinę funkcji:

$$y = \begin{cases} -1 & \text{gdy} \quad e < -\frac{\pi}{2} \\ \sin(\beta e) & \text{gdy} \quad -\frac{\pi}{2} \leq e \leq \frac{\pi}{2} \\ 1 & \text{gdy} \quad e > \frac{\pi}{2} \end{cases}$$

Taka postać funkcji jest szczególnie przydatna przy budowie sieci dokonującej transformaty Fouriera wejściowego sygnału. Sieć taką badał między innymi Lapedes [Lape87].

Niekiedy nieliniowość ma postać nie różniczkowalną, przydatną w praktycznych zastosowaniach, ale kłopotliwą do teoretycznej analizy. Z bardziej znanych nieliniowości tego typu wymienić trzeba:

— funkcję signum:

$$y = \begin{cases} 1 & \text{gdy} \quad e > 0 \\ 0 & \text{gdy} \quad e = 0 \\ -1 & \text{gdy} \quad e < 0 \end{cases}$$

— zmodyfikowaną funkcję signum:

$$y = \begin{cases} 1 & \text{gdy} \quad e > 0 \\ -1 & \text{gdy} \quad e \leq 0 \end{cases}$$

— funkcję skoku jednostkowego:

$$y = \begin{cases} 1 & \text{gdy} \quad e > 0 \\ 0 & \text{gdy} \quad e \leq 0 \end{cases}$$

— funkcję perceptronową:

$$y = \begin{cases} e & \text{gdy} \quad e > 0 \\ 0 & \text{gdy} \quad e \leq 0 \end{cases}$$

— funkcję BAM *(Bidirectional Associative Memory)*:

$$y^{(j+1)} = \begin{cases} 1 & \text{gdy} \quad e > 0 \\ y^{(j)} & \text{gdy} \quad e = 0 \\ -1 & \text{gdy} \quad e < 0 \end{cases}$$

— funkję BSB *(Brain State in a Box)*:

$$y = \begin{cases} 1 & \text{gdy} \quad e > 1 \\ e & \text{gdy} \quad 1 > e > -1 \\ -1 & \text{gdy} \quad e < -1 \end{cases}$$

— funkcję SPR *(Spatio-Temporal Pattern Recognition)*

$$y^{(j+1)} = y^{(j)} + A\left[-\alpha y^{(j)} + \beta e^{+}\right]$$

gdzie „funkcja ataku" $A[.]$ definiowana jest w sposób następujący:

$$A[u] = \begin{cases} u & \text{gdy} \quad u > 0 \\ \gamma u & \text{gdy} \quad u \leq 0 \end{cases}$$

a zapis e^{+} oznacza nieujemną wartość e:

$$e^{+} = \begin{cases} e & \text{gdy} \quad e > 0 \\ 0 & \text{gdy} \quad e \leq 0 \end{cases}$$

Podane wyżej funkcje wywodzą się głównie z publikacji inżynierów stosujących sieci neuronowe jako wygodne nowoczesne narzędzia przetwarzania informacji i poszukujących takich form opisu elementarnego procesora (jakim jest dla nich neuron), które pozwalają na wygodną analizę matematyczną zachodzących zjawisk (sigmoida), łatwą realizację techniczną (perceptron) lub wygodne modelowanie w formie programu symulacyjnego (signum). Tymczasem odmienne formy nieliniowości rozważają neurofizjologowie zajmujący się opisem neuronu jako obiektu biologicznego będącego przedmiotem ich badań. Tutaj chętnie stosuje się w różnych kontekstach funkcję logarytmiczną (nawiązującą do znanego prawa Webera — Fechnera). Oczywiście tę listę można by dowolnie wydłużać.

4.5 Uczenie nieliniowego neuronu

Rozważmy problem uczenia nieliniowych sieci neuronowych. Dla uproszczenia przyjmijmy, że analizować będziemy wyłącznie **regułę DELTA** w jej podstawowej postaci. Wzór opisujący zmianę wartości składowych wektora wag podczas prezentacji próbek z ciągu uczącego

$$U = << \mathbf{X}^{(1)}, z^{(1)} >, < \mathbf{X}^{(2)}, z^{(2)} >, \ldots, < \mathbf{X}^{(N)}, z^{(N)} >>$$

wygodnie będzie wyprowadzić początkowo dla jednego neuronu. Formuły uczenia szukać będziemy opierając się na wprowadzonej wyżej (przy rozważaniu sieci Adaline i Madaline) regule minimalizacji funkcjonalu błędu średniokwadratowego:

$$Q = \frac{1}{2} \sum_{j=1}^{N} \left(z^{(j)} - y^{(j)}\right)^2,$$

gdzie oczywiście

$$y^{(j)} = \varphi\left(\sum_{i=0}^{n} w_i^{(j)} x_i^{(j)}\right).$$

Rozkładając funkcjonał błędu na elementy składowe związane z poszczególnymi krokami procesu uczenia

$$Q = \sum_{(j=1)}^{N} Q^{(j)},$$

gdzie

$$Q^{(j)} = \frac{1}{2}\left(z^{(j)} - y^{(j)}\right)^2$$

możemy zgodnie z gradientową strategią procesu uczenia zapisać algorytm zmian współczynników wag

$$w_i^{(j+1)} - w_i^{(j)} = \Delta w_i^{(j)} = -\eta \frac{\partial Q^{(j)}}{\partial w_i}$$

Analogiczny wzór wprowadzono wcześniej dla sieci ADALINE, jednak treść tego wzoru jest w tym wypadku bogatsza, ponieważ obecnie zależność sygnału y od współczynników wag w_i ma uwikłany charakter, uwzględniający nieliniową funkcję $\varphi(e)$:

$$\frac{\partial Q^{(j)}}{\partial w_i} = \frac{\partial Q^{(j)}}{\partial y^{(j)}} \frac{\partial y^{(j)}}{\partial w_1} = \frac{\partial Q^{(j)}}{\partial y^{(j)}} \frac{dy^{(j)}}{de^{(j)}} \frac{\partial e^{(j)}}{\partial w_i}$$

Pierwszą pochodną występującą w omawianym wzorze można obliczyć bardzo łatwo

$$\frac{\partial Q^{(j)}}{\partial y^{(j)}} = -(z^{(j)} - y^{(j)}) = -\delta^{(j)},$$

podobnie jak trywialną postać ma także ostatni z rozważanych czynników

$$\frac{\partial e^{(j)}}{\partial w_i} = x_i^{(j)}.$$

Jedyny problem może powstać z czynnikiem $\frac{dy^{(j)}}{de^{(j)}}$ ponieważ oczywiście

$$\frac{dy^{(j)}}{de^{(j)}} = \frac{d\varphi(e)}{de^{(j)}},$$

a funkcja $\varphi(e)$ nie zawsze jest różniczkowalna. Dlatego tak chętnie (i trochę bezkrytycznie) przyjmowana jest w rozważaniach nad sieciami neuronowymi funkcja logistyczna

$$y = \varphi(e) = \frac{1}{1 + \exp(-\beta e)}$$

ponieważ zaletą tej funkcji jest prosta i łatwa do obliczenia wartość jej pochodnej

$$\frac{d\varphi(e)}{de^{(j)}} = y^{(j)}(1 - y^{(j)}).$$

Ostateczny wzór, na podstawie którego dokonuje się procesu uczenia sieci złożonej z nieliniowych elementów, może być zapisany w postaci

$$\Delta w_i^{(j)} = \eta \delta^{(j)} \frac{d\varphi(e)}{de^{(j)}} x_i^{(j)}$$

Wzór ten dla funkcji logistycznej zapisany może być w prostszej postaci

$$\Delta w_i^{(j)} = \eta (z^{(j)} - y^{(j)})(1 - y^{(j)}) x_i^{(j)} y^{(j)}$$

bezpośrednio użytecznej przy konstrukcji elementów naśladujących sieci neuronowe lub algorytmów symulujących te sieci.

4.6 Uczenie sieci nieliniowej

Opisany wyżej algorytm uczenia jest możliwy do bezpośredniego zastosowania jedynie w przypadku sieci jednowarstwowej. Dla sieci wielowarstwowych, o których wyżej stwierdzono, że mają istotnie szersze możliwości przetwarzania informacji, niż sieci jednowarstwowe, podany wyżej wzór nie daje się zastosować, ponieważ dla wewnętrznych warstw sieci nie ma możliwości bezpośredniego określenia oczekiwanych (wymaganych) wartości sygnałów wyjściowych $z^{(j)}$, a tym samym nie ma możliwości określenia błędu $\delta^{(j)}$. Istotnie, rozważając ciąg uczący w postaci

$$U = << X^{(1)}, Z^{(1)} >, < X^{(2)}, Z^{(2)} >, \ldots, < X^{(M)}, Z^{(M)} >>$$

mamy do dyspozycji n-wymiarowe wektory wejściowe $\mathbf{X}$ oraz k-wymiarowe wektory wyjściowe $\mathbf{Z}$ — stanowiące zestawy wymaganych sygnałów wyjściowych z elementów terminalnych czyli domykających sieć. Jeśli wektor $Y^{(j)}$ sygnałów wyjściowych z tych elementów nie będzie odpowiadal wymaganiom, to znaczy jeśli odnotujemy błąd wyrażający się różnicą wektorów $\mathbf{Z}^{(j)} - \mathbf{Y}^{(j)}$, to nie będziemy w stanie ustalić, w jakim stopniu za pojawienie się tego błędu odpowiadają neurony warstwy wyjściowej, na których ten błąd się ujawnił, a w jakim stopniu błąd powstal w elementach wcześniejszej warstwy, których sygnaly podawane byly jako wejściowe do neuronów ocenianej warstwy.

Przez wiele lat nie znano metody skutecznego uczenia sieci wielowarstwowych, a warstwy, dla których nie można bylo wyznaczyć sygnalu błędu znane byly w literaturze pod nazwą „warstw ukrytych" *(hiden layers)*. Dopiero w polowie lat 80-tych zaproponowany zostal algorytm wstecznej propagacji błędów *(backpropagation)* [Rume86], polegający na tym, że mając wyznaczony błąd $\delta^{(m)(j)}$ występujący podczas realizacji j-tego kroku procesu uczenia w neuronie o numerze m — możemy „rzutować" ten błąd wstecz do wszystkich tych neuronów, których sygnaly stanowily wejścia dla m-tego neuronu.

Opiszemy dokladniej ten ważny algorytm. Zacznijmy od sprawy oznaczeń. W sieci wielowarstwowej niemożliwe jest ścisle rozgraniczenie sygnalów wejściowych i wyjściowych, ponieważ sygnaly wyjściowe z neuronów wcześniejszej warstwy stają się zwykle sygnalami wejściowymi dla neuronów następnej warstwy. Z tego względu wprowadzimy jednolitą numerację wszystkich neuronów (bez dzielenia ich na wejściowe, wyjściowe i należące do warstw ukrytych) oraz zastosujemy stale oznaczenie $y_m^{(j)}$ dla sygnalu pojawiającego się na wyjściu m-tego neuronu w j-tym kroku procesu uczenia. Oczywiście niektóre spośród tych $y_m^{(j)}$ reprezentować będą sygnaly wyjściowe z calej sieci, możliwe do skonfrontowania z odpowiednimi skladowymi wektora $\mathbf{Z}^{(j)}$ w celu stwierdzenia, czy sieć pracuje poprawnie, czy też ma miejsce błąd. Podobnie uznamy, że sygnaly wejściowe, dotychczas oznaczane $\mathbf{X}$, będą także rozpatrywane jako sygnaly postaci $y_m^{(j)}$, czyli sygnaly występujące na wyjściach wyróżnionej grupy neuronów. Uprzednio (przy omawianiu pojedynczych neuronów lub sieci typu MADALINE) grupa ta nie istniala, a sygnal zewnętrzny docieral bezpośrednio do synaps przetwarzających informacje komórek. Teraz, wprowadzając dla sygnalów wejściowych także notację postaci $y_m^{(j)}$ zakladamy, jak gdyby niejawanie, że każde wejście „buforowane" jest przez neuron, którego zadaniem jest przetwarzanie wejściowego sygnalu $x_i^{(j)} \in \mathbf{X}^{(j)}$ na sygnal $y_m^{(j)}$, przy czym oczywiście neuron ten ma liniową charakterystykę i współczynnik wagi jedynego wejścia wynoszący 1, w wyniku czego

$$y_m^{(j)} = x_i^{(j)}.$$

Jako ciekawostkę warto odnotować fakt, że takie neurony „buforujące" sygnały wejściowe występują także w rzeczywistym systemie nerwowym, w szczególności w narządach zmysłów i w systemach percepcyjnych, gdzie są znane pod nazwą „komórek dwubiegunowych" (jako że mają dwa bieguny: jedno wejście i jedno wyjście). Bardziej szczegółowa dyskusja tego zagadnienia zawarta jest w pracy [Tade91a].

Jak z tego wynika wszystkie sygnały w rozważanej sieci mają postać $y_m^{(j)}$ i jedynie na podstawie numeru neuronu m można będzie rozróżnić, z jakim sygnałem (wejściowym, wyjściowym lub może jednym z sygnałów warstwy ukrytej) mamy w tym wypadku do czynienia. Dla skrócenia zapisów i zwiększenia czytelności dalszych rozważań wprowadzimy jednak wygodną notację opartą na wyróżnieniu w zbiorze numerów neuronów $\mathfrak{M}(\forall_m\ m \in \mathfrak{M})$ następujących podzbiorów:

- $\mathfrak{M}_x$ - zbiór numerów neuronów wejściowych do sieci;
- $\mathfrak{M}_y$ - zbiór numerów neuronów wyjściowych z sieci;
- $\mathfrak{M}_u$ - zbiór numerów neuronów warstwy ukrytej sieci;
- $\mathfrak{M}_i$ - zbiór numerów neuronów dostarczających sygnałów wejściowych do konkretnego, rozważanego aktualnie neuronu;
- $\mathfrak{M}_0$ - zbiór numerów neuronów do których dany, rozważany aktualnie neuron, wysyła swój sygnał wyjściowy.

Korzystając z tych oznaczeń możemy teraz wprowadzić zasadę uczenia wielowarstwowej sieci z użyciem algortmu *backpropagation*. Zaczniemy od stwierdzenia, że w j-tym kroku procesu uczenia sygnał na wyjściu każdego m-tego neuronu sieci może być wyznaczony z zestawu zależności o ogólnej postaci

$$y_m^{(j)} = \varphi\Big(\sum_{i \in \mathfrak{M}_i} w_i^{(m)(j)}\, y_i^{(j)}\Big),$$

gdzie przez $w_i^{(m)(j)}$ oznaczono wartość współczynnika wagowego synapsy łączącej wejście m-tego neuronu z wyjściem i-tego neuronu — oczywiście podczas j-tego kroku procesu uczenia. Podany wyżej wzór jest potencjalnie niejednoznaczny, ponieważ symbole typu $y_i^{(j)}$ występują w nim po lewej i po prawej stronie rozważanej formuły, można więc wyobrazić sobie sytuację, że dla pewnych i_1 i i_2 dla wyznaczenia $y_{i_1}^{(j)}$ potrzebne będzie $y_{i_2}^{(j)}$ i z kolei dla obliczenia wartości $y_{i_2}^{(j)}$ potrzebne będzie $y_{i_1}^{(j)}$. Jest to jednak niejednoznaczność pozorna, gdyż przy sieciach heteroasocjacyjnych *(feedforward)* zawsze można wyznaczyć taką kolejność obliczania wartości $y_m^{(j)}$, która gwarantuje, że w momencie wyliczania wartości $y_m^{(j)}$ dla kolejnego neuronu wszystkie wartości $y_i^{(j)}$ dla $i \in \mathfrak{M}_i$ będą już znane. Sprawa nie jest specjalnie trudna, wystarczy tylko kolejność obliczeń związać z kolejnością przekazywania sygnałów z wejścia sieci do neuronów w kolejnych warstwach: najpierw oblicza się $y_m^{(j)}$ dla $m \in \mathfrak{M}_x$, potem dla $m \in \mathfrak{M}_u$ (kolejno od wejścia w kierunku wyjścia), wreszcie na końcu obliczane są wartości dla $m \in \mathfrak{M}_y$. Sprawa się oczywiście komplikuje, jeśli w sieci występują sprzężenia zwrotne, jednak ten przypadek będzie rozważany osobno i chwilowo jest pominięty.

Określony porządek jest także wymagany przy znajdowaniu wartości poprawionych wag synaptycznych, z tym, że jest on przy metodzie *backpropagation* dokładnie przeciwny do tego, jaki jest wymagany przy obliczaniu wartości sygnałów $y_m^{(j)}$ w kolejnych elementach sieci.

Na samym początku wyznacza się zatem poprawki dla neuronów stanowiących wyjściową warstwę sieci ($m \in \mathfrak{M}_y$). Tu sprawa jest prosta, ponieważ dla poszczególnych sygnałów $y_m^{(j)}$ istnieją w ciągu uczącym wzorcowe (oczekiwane) wartości $z_m^{(j)}$, z którymi można je porównywać, wyznaczając bezpośrednio błąd $\delta_m^{(j)}$. Zakładamy przy tym dla prostoty, że numeracja składowych wektora wzorców $Z^{(j)}$ jest identyczna z numeracją neuronów tworzących wyjściową wastwę sieci (tylko przy zachowaniu tego warunku m w $y_m^{(j)}$ i m w $z_m^{(j)}$ mogą być utożsamiane!). Wówczas

$$\Delta w_i^{(m)(j)} = \eta\, \delta_m^{(j)} \frac{d\varphi(e)}{de_m^{(j)}}\, y_i^{(j)},$$

gdzie dla $m \in \mathfrak{M}_y$

$$\delta_m^{(j)} = z_m^{(j)} - y_m^{(j)}.$$

Wzór ten dla $m \in \mathfrak{M}_y$ oraz dla $\varphi(e)$ w postaci funkcji logistycznej zapisany może być w postaci

$$\Delta w_i^{(m)(j)} = \eta\, (z_m^{(j)} - y_m^{(j)})\, (1 - y_m^{(j)})\, y_i^{(j)} y_m^{(j)}$$

dogodnej do praktycznych obliczeń.

Na razie wprowadzane wzory nie wnoszą istotnej nowości; ich zapis jest trochę bardziej złożony (przez konieczność uwzględniania numerów m), jednak ich ogólna struktura jest identyczna z wcześniej wprowadzonymi wzorami opisującymi proces uczenia dla pojedynczego neuronu. Jednak z chwilą skoncentrowania uwagi na neuronach warstw ukrytych sytuacja się zmienia. Można bowiem przez analogię zapisać także dla tych neuronów regułę

$$\Delta w_i^{(m)(j)} = \eta\, \delta_m^{(j)} \frac{d\varphi(e)}{de_m^{(j)}}\, y_i^{(j)}$$

jednak dla $m \in \mathfrak{M}_u$ nie ma możliwości bezpośredniego określenia wartości $\delta_m^{(j)}$. Rozważmy jednak zbiór $\mathfrak{M}_0$. Zgodnie z wprowadzonymi oznaczeniami jest to zbiór neuronów, do których dociera sygnał $y_m^{(j)}$ czyli sygnal wyjściowy z rozważanego neuronu warstwy ukrytej. Załóżmy na chwilę, że $\mathcal{M}_0 \subseteq \mathcal{M}_y$, to znaczy załóżmy, że rozważany neuron należy wprawdzie do warstwy ukrytej, ale jego sygnal wyjściowy dociera wyłącznie do neuronów warstwy wyjściowej, czyli tych, dla których wartości błędów $\delta_k^{(j)}$ mogą być bez trudu określone. Przy tym założeniu można wykazać [Rume86], że błąd $\delta_m^{(j)}$ neuronu warstwy ukrytej może być obliczony poprzez wsteczne rzutowanie (czyli *backpropagation*) błędów wykrytych w warstwie odbierającej sygnały:

$$\delta_m^{(j)} = \sum_{k \in \mathfrak{M}_0} w_m^{(k)(j)}\, \delta_k^{(j)}$$

Warto zwrócić uwagę na współczynniki wagowe, wykorzystywane przy wstecznym rzutowaniu błędów. Czynnik $w_m^{(k)(j)}$ należy rozumieć jako wagę występującą w neuronie o numerze k przy jego wejściu numerze m, czyli odbierającym sygnal od aktualnie rozważanego neuronu. Oznacza to, że rzutowane wstecznie błędy mnożone są przez te same współczynniki, przez które mnożone były przesyłane sygnały, tyle tylko, że kierunek przesyłania informacji zostaje w tym wypadku odwrócony: zamiast od wejścia do wyjścia przesyła się je od wyjścia kolejno w kierunku wejścia.

Stosując opisaną wyżej technikę wstecznej propagacji błędów można nauczyć całą sieć, ponieważ każdy neuron warstwy ukrytej albo znajduje się w przedostatniej warstwie sieci

i przesyla swoje sygnaly do neuronów wyjściowych (wtedy jego błąd może być wyznaczony wyżej podaną metodą) albo znajduje się w jednej z głębiej ukrytych warstw, podając sygnaly do neuronów innych warstw ukrytych — wtedy jego błąd można oszacować z chwilą obliczenia błędów w neuronach będących odbiorcami jego sygnalów. W sumie jednak w sieci *feedforward* zawsze da się wyznaczyć taką kolejność wstecznej propagacji błędów, która pozwoli obliczyć błędy $\delta_m^{(j)}$ dla wszystkich elementów sieci, a tym samym pozwoli znaleźć wszystkie wartości $w_i^{(m)(j+1)}$ na podstawie wartości $w_i^{(m)(j)}$ i występujących w sieci sygnalów. Doświadczenie wskazuje, że uczenie wykonywane tą metodą jest stosunkowo skuteczne, ale bardzo powolne. Jego przyspieszenie próbujemy uzyskać stosując opisane dla sieci MADALINE techniki momentum, kumulacji błędów, odrzucania malych poprawek itp. Wyniki bywają różne: w niektórych zadaniach udaje się przyspieszyć proces uczenia nawet kilkadziesiąt razy, w innych prowadzi to do braku zbieżności.

4.7 Dobór parametrów uczenia sieci

Skoro mowa o zbieżności warto dodać jedną uwagę. Bardzo ważne jest wlaściwe dobieranie współczynnika η w taki sposób, by nie utracić możliwości precyzyjnego kontrolowania procesu uczenia. Większość danych litraturowych wskazuje na celowość stosowania większych wartości η (i wszystkich pokrewnych współczynników, na przyklad przy skladniku mementum) na początku uczenia i zmniejszania ich w miarę, jak współczynniki wagowe zbliżają się do swoich ustalonych wartości. Jednak obok tej strategii dobre wyniki daje także strategia bardziej zlożona, zakladająca bardzo male wartości η (i ewentualnie innych współczynników) na początku procesu uczenia, zwiększane potem do pewnych wartości znamionowych w momencie kiedy proces uczenia jest już zaawansowany, a następnie już w klasyczny sposób zmniejszane stopniowo w końcowym okresie uczenia. Tego typu postępowanie jest uzasadnione faktem, że w pierwszych krokach procesu uczenia następuje wstępna preorientacja neuronów, których wagi, początkowo ustawione losowo i na ogól bliskie zera, zostają z wolna spolaryzowane w kierunku pobudzających lub hamujących połączeń o decydującym znaczeniu dla dalszego toku procesu uczenia. W trakcie tego subtelnego wybierania początkowych odcinków trajektorii wiodących do ostatecznego rozpoznania bardzo latwo o pomylki, będące wynikiem przypadkowego skladu obiektów ciągu uczącego, zaś korekta tych pomylek jest latwiejsza, jeśli proces uczenia będzie początkowo przebiegal w tempie celowo zwolnionym. Potem uczenie może nabrać szybkości, by ponownie zwolnić pod koniec procesu, gdy konieczne są już tylko kosmetyczne korekty ustalanych wektorów wag.

W literaturze jest mnóstwo opisów udanych eksperymentów uzyskania poprawnego dzialania nauczonej sieci po zastosowaniu algorytmu *backpropagation*. Tytulem przykladu wspomnijmy o badaniach prowadzonych przez Gale L. Martina i Jamensa A. Pittmana opisanych w pracy [Mart91]. W pracy tej rozpoznawano ręcznie pisane znaki (litery i cyfry) mające znormalizowane rozmiary 15 × 24 piksele (o skali szarości zmienianej w sposób ciągly na odcinku [0, 1]) posługując się zbiorem uczącym liczącym 35 200 cyfr i 6 300 liter pisanych przez 110 ludzi. Zbiory testowe tworzylo odpowiednio 4 000 cyfr i 2 368 liter. Test rozpoznawania zniekształconych (procesem wprowadzania do komputera) znaków przez ludzi wykazal, że popelniali oni średnio 3,4% błędów podczas rozpoznawania. Nauczona sieć popelniala około[4]. 3% błędów, co można uznać za bardzo dobry wynik. Podczas uczenia korzystano z

[4] Wielkość błędu popełnianego przez sieć zależy od tego, czy sygnały wyjściowej warstwy sieci traktuje się jako jednoznaczne wskazania odpowiednich obrazów, czy też dopuszcza się brak odpowiedzi (gdy wielkość

klasycznej metody backpropagation z wykorzystniem elementu momentum i przyjmowano współczynniki uczenia wynoszące odpowiednio $\eta_1 = 0,05$ i $\eta_2 = 0,9$.

Bardzo interesujące są przytoczone w cytowanej wyżej pracy wyniki dotyczące pojemności pamięci sieci i jej zdolności do uogólniania informacji otrzymywanych w trakcie procesu uczenia. Wcześniej przytaczane wyniki na temat zdolności sieci do generalizacji wiadomości uzyskiwanych w trakcie procesu uczenia [Baum89] wskazywały na konieczność zmniejszenia liczby warstw, liczby połączeń i liczby bitów przeznaczonych na zapamiętanie wagi jednego połaczenia. Podobne wnioski prezentuje praca [LeCu89], w której wskazano, że przy zadaniu rozpoznawania ręcznie pisanych liter i cyfr sieć zostaje „zmuszona" do skutecznej generalizacji po przyjęciu ograniczenia rozmiarów warstwy ukrytej do 40 neuronów. Cytowana praca [Mart91] nie potwierdza tych przypuszczeń. Opisana w niej sieć zachowywała się bardzo podobnie i uzyskiwała zbliżone poziomy błędów po nauczeniu przy różnych liczbach elementów w warstwie ukrytej — od 50 do 383 elementów.

Próbowno też różnych zasad łączenia elementów sieci ze sobą: Stosowano zasadę globalnego łączenia na zasadzie „każdy z każdym" pomiędzy wejściową „siatkówką" (15x24), a pierwszą warstwą ukrytą (150 neuronów) oraz pomiędzy pierwszą warstwą ukrytą i drugą warstwą (50 neuronów) przeciwstawiając ją zasadzie połączeń lokalnych. Połącznia lokalne wytworzono na dwa sposoby. W pierwszym z nich 540 neuronów pierwszej warstwy ukrytej ma połączenia jedynie z podobszarami „siatkówki" o rozmiarach 5x8 pikseli, rozmieszczonymi w taki sposób, że się częściowo pokrywają (przesunięcie środka okienka dokonuje się na szerokości 2 pikseli z każdej strony). Powstaje w ten sposób struktura 6x9 okienek i do każdego z tych okienek dołączonych jest 10 neuronów, wyznaczających — jak to w warstwie ukrytej — pewne cechy wejściowych (lokalnych) fragmentów obrazu. Te cechy są z kolei kodowane przez 100 neuronów drugiej warstwy ukrytej, które to neurony mają połączenia na zasadzie „każdy z każdym" z 540 neuronami pierwszej warstwy.

Druga struktura lokalna ma bardziej wyrafinowany charakter i zakłada grupowanie neuronów pierwszej warstwy ukrytej (zgodnie z sugestiami LeCuna [LeCu89] określanymi w literaturze jako koncepcja *shared weights*). W tej drugiej rozważanej strukturze neurony pierwszej warstwy ukrytej są nadal związane z lokalnymi podobszarami obrazu o rozmiarach 5x8 pikseli, jednak tworzą one uklad „sześcianu" o rozmiarach 6x9x10 neuronów, przy czym każdy z neuronów wchodzących w sklad jednej „warstwy" sześcianu (6x9) jest uczony w powiązaniu z pozostalymi elementami tej samej warstwy. W efekcie wszystkie elementy danej warstwy uzyskują te same wartości wag, co ma taką interpretację, że powinny one wykrywać pewne powtarzające się lokalne cechy obrazu *niezależnie od ich lokalizacji na rastrze*. Z kolei druga warstwa ukryta sklada się z 102 neuronów tworzących 17 grup po 6 neuronów. Neurony każdej grupy komunikują się wyłącznie z neuronami wybranego fragmentu wspomnianego wyżej „sześcianu" neuronów pierwszej warstwy, przy czym rozmiary wszystkich fragmentów wynoszą 4x5x10, a ich przesunięcie wynosi 2. Wśród tych wszystkich szczegółów nie można zagubić najważniejszego elementu: interpretacji poczynionych założeń. Otóż wprowadzenie lokalnych połączeń i grupowania neuronów zmierza generalnie do ograniczenia wielkości pamięci sieci (reprezentowanej przez liczbę niezależnie ustawianych wag).

Mimo bardzo wyrafinowanych kombinacji, wiążących rozmiar i strukturę sieci z rozwią-

wyjściowego sygnalu elementu „zwyciężającego" wszystkie pozostale jest zbyt mala). I tak przy rozpoznawaniu cyfr odnotowano błąd 4% przy calkowitym wykluczeniu odpowiedzi odmownych, ale błąd zmalal do 3% przy dopuszeniu 5% odpowiedzi odmownych i spadł poniżej 1% przy dopuszczeniu 10% odpowiedzi odmownych.

zywanym zadaniem rozpoznawania liter i cyfr — w referowanych badaniach nie stwierdzono istotnego związku pomiędzy szybkością uczenia, a rozmiarem czy topologią sieci, jak również nie wykryto znamiennego związku pomiędzy topologią sieci, a uzyskaną jakością realizacji stawianych przed siecią zadań (tu — liczbą błędów przy rozpoznawaniu zbioru testowego).

Rozdział 5

Sieci CP

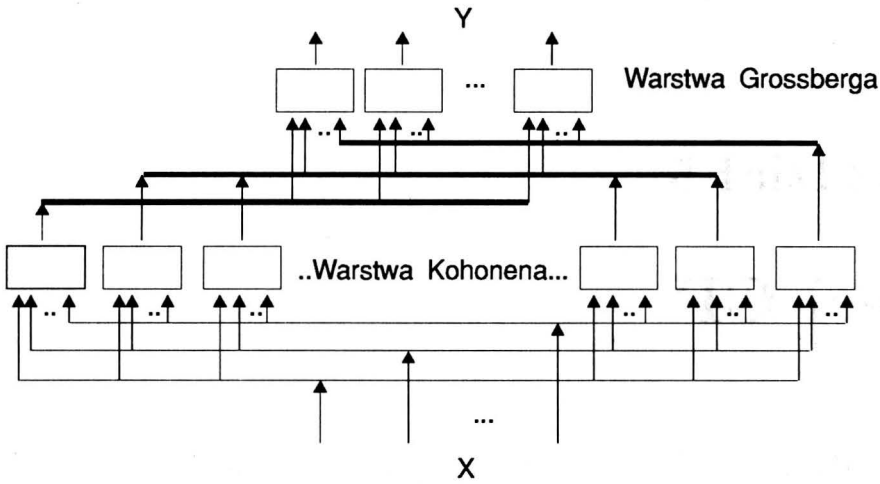
5.1 Pomysł sieci „przesyłającej żetony" (Hecht-Nielsena)

Wada, jaką jest powolny i uciążliwie pracochlonny proces uczenia w sieciach ze wsteczną propagacją błędów sprawila, że w literaturze dotyczącej sieci neuronowych pojawilo się wiele propozycji sieci o podobnym przeznaczeniu, lecz sprawniejszych w dzialaniu. Jedną z najbardziej znanych jest propozycja Roberta Hecht-Nielsena [Hech88], określana jako sieć *Counter-Propagation*. Nazwa ta jest trudna do przetlumaczenia na język polski, gdyż tytulowy *counter* ma wiele znaczeń. Ponieważ najlepiej przystającym polskim terminem wydaje się być „żeton" (używany w banku do identyfikacji klienta w trakcie większych operacji kasowych), dlatego zaproponowano nazwę „sieć przesylająca żetony" z pelną świadomością jej niedoskonalości i z nadzieją, że ktoś z Czytelników książki zaproponuje lepszą nazwę. W dalszych rozważaniach używać będziemy oznaczenia CP dla określenia tej sieci.

Sieć CP wlaściwie nie jest oryginalną propozycją, lecz stanowi kompilację sieci Kohonena i sieci Grossberga. Jednak zestawienie tych sieci w strukturze sieci CP wprowadziło istotnie nową jakość - sieć stosunkowo szybko się uczącą i mającą (potencjalnie) nieograniczony zakres możliwych odzworowań pomiędzy sygnalem wejściowym **X** i wyjściowym **Y**.

Ze względu na dość prostą zasadę działania, funkcjonowanie sieci CP porównywane jest do odczytu gotowego wyniku z tablicy (tzw. technika *look-up table*, chętnie stosowana przy programowaniu sterowników przemysłowych). Oczywiście sieć CP nie ogranicza swojego działania do odczytu z tablicy, gdyż wejściowa jej warstwa dokonuje dodatkowo adaptacyjnej klasyfikacji wejściowych bodźców, co pozwala na pewne uogólnianie gromadzonego doświadczenia, a ponadto jak wszystkie sieci neuronowe CP dysponuje możliwością uczenia, co pozwala na jej wygodne używanie w szerokim zakresie ciekawych zastosowań.

Na podanym niżej rysunku połączenia między warstwą Kohonena, a warstwą wyjściową narysowano w uproszczeniu. W rzeczywistości mamy tu do czynienia z połączeniem według reguły „każdy z każdym", co jest jednak trudne do narysowania (i do realizacji...), ponieważ z reguły liczba neuronów w warstwie Kohonena jest bardzo duża (im większa, tym bogatsze możliwości sieci!); jak zaznaczono na rysunku, jest to liczba znacznie większa, niż liczba składowych wektora wejść i wektora wyjść z sieci.



5.2 Działanie pierwszej warstwy sieci CP

Podstawowym założeniem przy stosowaniu sieci CP jest normalizacja sygnałów wejściowych. Każdy wektor $\mathbf{X}$ wprowadzany do systemu musi spełniać warunek

$$\|\mathbf{X}\| = 1$$

który można rozpisać w formie

$$\mathbf{X}^T \mathbf{X} = 1$$

Oczywiście normalizacja $\mathbf{X}$, dokonywana według reguły

$$x_i' = \frac{x_i}{\sum_{j=1}^{n} x_j}$$

musi być wykonywana poza siecią, gdyż jako operacja nieliniowa sprawia poważne kłopoty przy próbie realizacji „czysto neuronowej".

Normalizacja wejść jest potrzebna ze względu na element konkurencji (rywalizacji) występujący w pierwszej warstwie sieci CP. Do dalszego przetwarzania w kolejnej warstwie sieci przesyłany jest wyłącznie jeden sygnał („żeton"), pochodzący od tego elementu pierwszej warstwy, który jest optymalnie dopasowany do przedstawionego sygnału wejściowego $\mathbf{X}$. Pierwsza warstwa sieci CP jest warstwą realizującą algorytm Kohonena. Oznacza to, że wejściowe wektory $\mathbf{X}$ mnożone są przez wektory wag $\mathbf{W}_j$ poszczególnych neuronów sieci dostarczając wartości e_j będących sumarycznym (ważonym) pobudzeniem każdego neuronu,

$$e_j = \boldsymbol{W}_j^T \boldsymbol{X}$$

a następnie wybierany jest element o największej wartości pobudzenia e_j („zwycięzca") i tylko jego sygnał wyjściowy przyjmuje wartość 1

$$k_j = \begin{cases} 1 & \text{gdy } \forall_{i \neq j} e_j > e_i \\ 0 & \text{w przeciwnym przypadku} \end{cases}$$

Jest to właśnie tytulowy *counter* — żeton zastępujący i symbolizujący wszystkie sygnaly wejściowe podobnie jak żeton klienta w banku zastępuje i symbolizuje całą jego sprawę.

Zasada dzialania warstwy Kohonena zaklada, że sygnal e_j każdego neuronu jest miarą stopnia podobieństwa pomiędzy aktualnym sygnalem wejściowym $\mathbf{X}$, a abstrakcyjnym wzorcem sygnalu, na którego wykrywanie wytrenowany jest j-ty neuron. Ten wzorzec idealnego sygnalu dla j-tego neuronu zawarty jest w jego wektorze wag $\mathbf{W}_j$. Jeśli $\mathbf{X} = \mathbf{W}_j$, wówczas neuron odpowiada sygnalem o maksymalnej wartości, jeśli $\mathbf{X} \neq \mathbf{W}_j$, wówczas e_j jest miarą kosinusa kąta α pomiędzy wektorami $\mathbf{X}$ i $\mathbf{W}_j$

$$e_j = \mathbf{W}_j^T \mathbf{X} = \|\mathbf{W}_j^T\| \ \|\mathbf{X}\| \cos\alpha$$

Miara ta jest wiarygodna, jeśli $\|\mathbf{W}_j\| = 1$ i $\|\mathbf{X}\| = 1$, w przeciwnym razie możliwe są przykre pomylki. Szczególnie groźny jest przypadek dopuszczenia dowolnych wartości $\|\mathbf{W}_j\|$. Rozważmy trywialny przyklad.

5.3 Przykład sieci CP pokazujący wady jej działania

Niech sieć sklada się z dwóch neuronów o trzech wejściach każdy i niech wektory wag zarejestrowane w sieci będą następujące

$$\mathbf{W}_1 = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \qquad \mathbf{W}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

Wyobraźmy sobie, że na wejściu sieci pojawia się sygnal

$$\mathbf{X} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Sumaryczne pobudzenia neuronów wynoszą wtedy

$$e_1 = 14 \quad , \quad e_2 = 2$$

i oczywiście neuron numer 1 jest „zwycięzcą". Jest to absolutnie prawidlowe, ponieważ wektor wejściowy byl tu zgodny z wzorcem zapamiętanym przez pierwszy neuron. Rozważmy jednak, co się stanie po podaniu na wejście sieci sygnalu zgodnego ze wzorcem pamiętanym przez drugi neuron. Okazuje się, że dla

$$\mathbf{X} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

sumaryczne pobudzenia neuronów wynoszą

$$e_1 = 2 \quad , \quad e_2 = 1$$

i znowu neuron numer 1 jest „zwycięzcą". Jest to oczywisty bląd, a jego przyczyna związana jest z brakiem normalizacji sygnalów. Gdyby wejścia $\mathbf{X}$ byly normalizowane, wartości wektorów wag powstające w wyniku procesu uczenia (opisanego niżej) także spelnialy by warunek $\|\mathbf{W}_j\| = 1$ i do podobnych paradoksów nigdy by nie doszlo.

5.4 Zadania drugiej warstwy sieci

Sumarycznie pierwsza warstwa sieci może być opisana wzorem

$$\mathbf{K} = \Xi(\mathbf{W}\,\mathbf{X})$$

gdzie $\mathbf{W}$ jest macierzą wag, utworzoną z transponowanych wektorów $\mathbf{W}_j$ traktowanych jako kolejne wiersze, $\mathbf{X}$ jest wektorem **normalizowanych** sygnałów wejściowych, a nieliniowy operator $\Xi(o)$ odpowiedzialny jest za wybór „zwycięskiego" elementu warstwy Kohonena, ustawienie jego sygnału wyjściowego na wartość 1 i wyzerowanie wszystkich innych wyjść.

Warto zauważyć, że w praktycznie stosowanych sieciach klasy CP macierz $\mathbf{W}$ daleko odbiega od macierzy kwadratowej, gdyż liczba elementów w warstwie Hohonena przyjmowana jest zwykle jako znacznie większa, niż liczba składowych wektora $\mathbf{X}$. Te liczne elementy warstwy Kohonena mają wydobyć z wejściowego sygnału jego istotne cechy i wybrać stosowny „żeton" (jedyny element $k_i \in \mathbf{K}$ różny od zera), który będzie przekazany do następnej warstwy.

Druga warstwa sieci realizuje algorytm *Outstar* Grossberga. Sygnały wejściowe do tej warstwy tworzą wektor $\mathbf{K}$, a sygnał wyjściowy $\mathbf{Y}$ obliczany jest według klasycznej reguły

$$\mathbf{Y} = \mathbf{V}\,\mathbf{K}$$

gdzie macierz współczynników wagowych $\mathbf{V}$ składa się z transponowanych wektorów $\mathbf{V}_i$ odpowiadających zestawom wag kolejnych neuronów warstwy wyjściowej. Macierz $\mathbf{V}$ jest także daleka od kwadratowego kształtu, gdyż z reguły bardzo dużemu wymiarowi wektora $\mathbf{K}$ towarzyszy niewielki (porównywalny z wymiarem wektora $\mathbf{X}$) wymiar $\mathbf{Y}$.

Z formalnego punktu widzenia neurony warstwy wyjściowej mają dużo pracy, ponieważ muszą obliczać sumy

$$y_i = \sum_{j=1}^{m} v_{ij}\, k_j$$

gdzie m ma z reguły dużą wartość. W praktyce jednak jedynie jeden element wektora $\mathbf{K}$ ma wartość 1, a pozostałe mają wartość 0 i wystarcza utożsamienie wyjścia y_i z pewnym współczynnikiem wagowym v_{ij}. Zauważmy, że to działanie rzeczywiście przypomina odczyt z gotowej tabeli. Na każdym i-tym wyjściu sieci w trakcie procesu uczenia przygotowane zostaje m wariantów odpowiedzi v_{ij}. Gdy warstwa Kohonena ustali, które z jej licznych wyjść otrzyma „żeton" — na wszystkich wyjściach pojawiają się tylko te wartości v_{ij}, które odpowiadają numerowi j, dla którego $k_j = 1$.

5.5 Uczenie pierwszej warstwy sieci CP

Aby opisany wyżej schemat postępowania prowadził do sensownych wyników — konieczne jest poprawne ustawienie wszystkich wartości w_{ij} oraz v_{ij} czyli — realizacja procesu uczenia.

Uczenie sieci CP przebiega równocześnie w obydwu warstwach sieci, ale podczas omawiania wygodnie jest opisać oddzielnie, jak przebiega proces uczenia w pierwszej warstwie, a jak w drugiej. Uczenie całej sieci CP jest uczeniem z nauczycielem, gdyż ciąg uczący ma formę

$$\boldsymbol{U} = \{< \boldsymbol{X}^{(1)}, \boldsymbol{Z}^{(1)} >, < \boldsymbol{X}^{(2)}, \boldsymbol{Z}^{(2)} >, \ldots, < \boldsymbol{X}^{(k)}, \boldsymbol{Z}^{(k)} >, \ldots, < \boldsymbol{X}^{(N)}, \boldsymbol{Z}^{(N)} >\}$$

a zatem wraz z każdym wektorem wejściowym $\mathbf{X}$ podawany jest wektor wyjściowy, jaki użytkownik chce uzyskać z sieci. Jednak mimo tego założenia przy uczeniu nie wykorzystuje się pojęcia błędu (a więc nie jest to rodzaj algorytmu Delta), a ponadto mimo występowania dwóch warstw nie wykorzystuje się w żadnej formie informacji pochodzących od nauczyciela przy uczeniu pierwszej („ukrytej") warstwy sieci. Pomysl jest bardzo prosty i skuteczny: przy uczeniu pierwszej warstwy stosuje się technikę Kohonena, która jest formą uczenia bez nauczyciela. Natomiast przy uczeniu drugiej warstwy wykorzystuje się algorytm Grossberga do bezpośredniego wymuszania pożądanych odpowiedzi sieci. Rozważmy to teraz dokladniej.

Zaczniemy od opisu sposobu uczenia pierwszej warstwy sieci. Zgodnie z regulą Kohonena uczenie przebiega następująco. Na k-tym kroku pokazuje się wektor wejściowy $\mathbf{X}^{(k)}$, a dysponując (z wcześniejszych kroków procesu uczenia) wartościami wszystkich wektorów $\mathbf{W}_j^{(k)}$ można obliczyć wszystkie wartości $e_j^{(k)}$

$$\mathbf{e}_j^{(k)} = \mathbf{W}_j^{(k)T}\,\mathbf{X}^{(k)} \quad , \quad j = 1, 2, \ldots, m$$

oraz wyznaczyć numer „zwycięskiego" neuronu (tzn. tego, dla którego zachodzi)

$$\forall\,(j \neq z) \qquad e_z^{(k)} > e_j^{(k)}$$

Korekcie podlegają wyłącznie wagi „zwycięskiego" neuronu według reguły

$$\boldsymbol{W}_z^{(k+1)} = \boldsymbol{W}_z^{(k)} + \eta_1\,(\boldsymbol{X}^{(k)} - \boldsymbol{W}_z^{(k)})$$

Współczynnik uczenia η_1 przyjmowany jest zwykle jako równy 0,7 na początku procesu uczenia i stopniowo zmniejszany dla większych k.

Podobnie jak w wielu innych algorytmach samouczenia, przy realizacji metody Kohonena najważniejsze są pierwsze kroki, bo od nich w znacznym stopniu zależy powodzenie calej pracy. Najpierw trzeba nadać współczynnikom wagowym w_{ij} wartości początkowe. Nie jest to tak proste, jak przy niektórych innych metodach, ponieważ powinno się zapewnić unormowanie wszystkich początkowych wektorów wag

$$\|\boldsymbol{W}_j^{(1)}\| = 1$$

a ponadto wysoce pożądane jest takie dobranie ich kierunków, by w sposób równomierny rozkladaly się na powierzchni sfery jednostkowej w przestrzeni n-wymiarowej. Takie zainicjowanie jest jednak trudne i nie gwarantuje dobrych warunków do realizacji procesu samouczenia, gdyż jest możliwe pojawianie się w trakcie uczenia kilku „zwycięskich" neuronów podczas prezentacji uczących sygnalów $\mathbf{X}^{(k)}$, co utrudnia realizację opisanego wyżej procesu uczenia. Jedna z technik zapobiegania tym niekorzystnym zjawiskom, nazywana w literaturze *convex combination method*, polega na tym, że początkowo wszystkim składowym wszystkich wektorów wag nadaje się tę samą wartość początkową

$$w_{ij}^{(1)} = \sqrt{1/n}$$

Powoduje to, że wszystkie wektory $\mathbf{W}_j^{(1)}$ są prawidlowo unormowane, ale wszystkie pokrywają się. Po takim zainicjowaniu wektorów wag zaczyna się opisany wyżej proces uczenia, ale jako wektory wejściowe podaje się wektory o współrzędnych obliczanych według wzoru

$$x_i^{(k)'} = \eta_2\,(k)\,x_i^{(k)} + \left[1 - \eta_2\,(k)\right]\sqrt{1/n}$$

Funkcja adaptująca $\eta_2(k)$ dla małych k przyjmuje bardzo małe wartości, natomiast potem powoli rośnie do wartości 1 i tę wartość zachowuje przez resztę procesu uczenia. Sens takich zmian wektora $\mathbf{X}^{(k)}$ jest oczywisty: na początku poszczególne skorygowane wektory $\mathbf{X}^{(k)}$ bardzo mało różnią się od siebie, ale za to prawie dokładnie pokrywają się z wektorami wag $\mathbf{W}_j^{(k)}$. Wśród „zwycięskich" neuronów (początkowo są to wszystkie neurony wyjściowej warstwy, potem stopniowo jest ich coraz mniej) przeprowadza się prosty zabieg porządkujący, polegający na tym, że uznaje się za „zwycięski" neuron o najniższym numerze i tylko jego wagi poddaje się procesowi uczenia — też początkowo bardzo subtelnego, jak wynika ze wzoru opisującego sposób uczenia i przyjętej reguły modyfikacji wektorów $\mathbf{X}^{(k)}$. Dzięki opisanym zabiegom wektory $\mathbf{W}_j^{(k)}$ początkowo współliniowe, zaczynają się z wolna rozchodzić w kierunkach wynikających z naturalnych tendencji występujących w zbiorze wejściowych sygnałów $\mathbf{X}^{(k)}$, dając początek sprawnej i efektywnie dalej przebiegającej klasteryzacji. Warto podkreślić, że opisane zabiegi skutkują jedynie w krótkim początkowym okresie uczenia, gdyż wkrótce funkcja $\eta_2(k)$ osiąga wartość 1 i modyfikacja wektorów $\mathbf{X}^{(k)}$ praktycznie przestaje działać.

5.6 Uczenie drugiej warstwy sieci CP

W porównaniu z kłopotami, jakich nastręcza uczenie pierwszej warstwy sieci CP, uczenie warstwy Grossberga to sama przyjemność. Reguła zmiany wag v_{ij} podporządkowana jest tu zasadzie Widrowa-Hoffa:

$$v_{ij}^{(k+1)} = v_{ij}^{(k)} + \eta_3 \left(y_i - z_i\right) k_j$$

Reguła ta okazuje się jeszcze prostsza w stosowaniu, niż to może wynikać z podanego wzoru, gdyż tylko jedna wartość k_j jest różna od zera i w każdym kroku procesu uczenia korygowane są tylko te wagi, które łączą poszczególne neurony wyjściowej warstwy z jednym tylko — mianowicie „zwycięskim" elementem poprzedniej warstwy. Ta zasada (zwana czasem regułą *outstar*) znakomicie zmniejsza pracochłonność procesu uczenia.

Parametr η_3 wybiera się zazwyczaj ostrożnie tak, by proces uczenia nie spowodował wpisania do „tablicy" *(look-up table)* błędnych wartości. Zwykle zaczyna się uczenie od wartości $\eta_3 = 0,1$ a potem się tę wartość jeszcze mocniej redukuje.

5.7 Przykład zadania rozwiązywanego przez sieć CP

Sieci CP okazują się zaskakująco przydatne przy rozwiązywaniu różnych praktycznych zadań. Przykładowo można tu wspomnieć o modelu sieci CP wbudowanym w symulator NeuralWorks Professional II firmy NeuralWare. Model ten ma doradzać, jak spędzać sobotni wieczór.

Sieć ma dwa wejścia:

- x_1 określające ilość pilnej pracy, jaka jest do wykonania,
- x_2 podające „temperaturę" uczuć rodzinnych.

Sieć ma też pięć wyjść:

- y_1 - oznacza czytanie książki,
- y_2 - wspólne zakupy,
- y_3 - wspólny spacer w parku,
- y_4 - zabranie żony na wystawną kolację do restauracji,
- y_5 - pozostanie w pracy.

Sieć uczono za pomocą następujących przykladów:

$x_1 = 0;$	$x_2 = 0;$	$y_1 = 1$	(gdy nie ma co robić, a brak sympatii to najlepiej sobie poczytać);
$x_1 = 0,5;$	$x_2 = 0;$	$y_1 = 1$	(za mało pilnej pracy, żeby sobie psuć sobotnie popołudnie);
$x_1 = 0;$	$x_2 = 0,5;$	$y_2 = 1$	(uczuć rodzinnych wystarcza akurat na wspólne sobotnie zakupy);
$x_1 = 1;$	$x_2 = 1;$	$y_3 = 1$	(mimo nawału pracy da się wygospodarować czas na wspólny spacer w parku);
$x_1 = 0,5;$	$x_2 = 1;$	$y_4 = 1$	(najważniejsze są uczucia i romantyczna kolacja we dwoje);
$x_1 = 1;$	$x_2 = 0,5;$	$y_5 = 1$	(tę pilną pracę trzeba koniecznie wykonać).

Po treningu sieci przedstawiono zadanie, polegające na podaniu sygnalów

$$x_1 = 0; \qquad x_2 = 1$$

Warto zauważyć, że takiego przypadku nie bylo w ciągu uczącym! Sieć podala odpowiedź

$$y_4 = 1$$

Trudno nie uznać tej odpowiedzi za poprawną decyzję, mimo że pochodzi ona z sieci mającej zaledwie pięć neuronów w warstwie Kohonena i pięć w warstwie Grossberga.

5.8 Autoasocjacyjna sieć CP

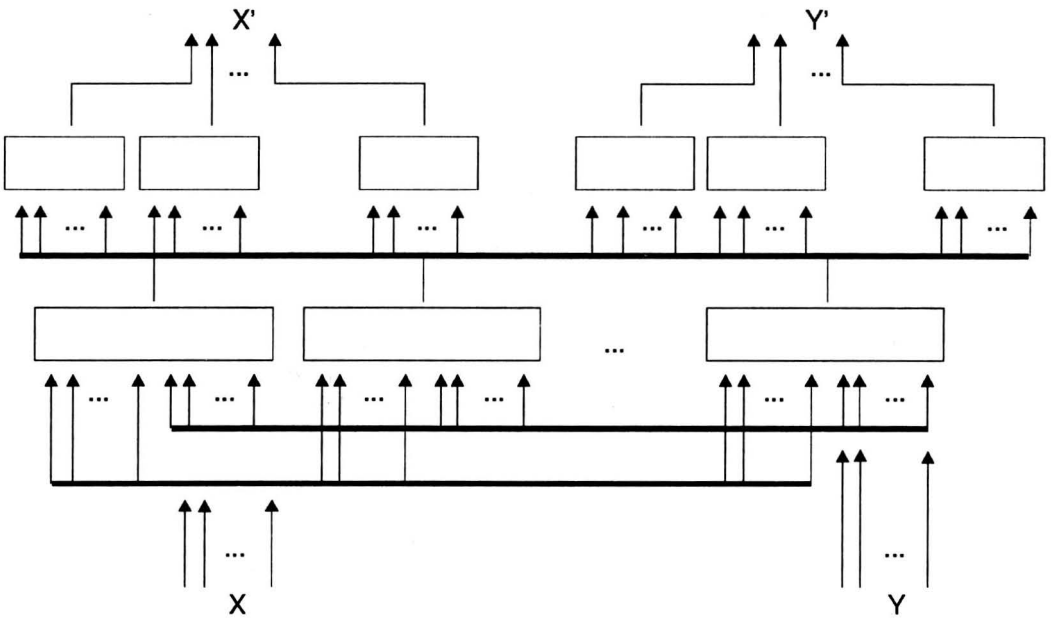
Doświadczenie to dowodzi, że sieć CP potrafi uogólniać i kojarzyć dostarczone jej informacje, co podnosi jej przydatność i sprawia, że jest ona (oczywiście w bardziej rozbudowanej wersji) chętnie i z powodzeniem stosowana. Jedną z bardziej interesujących możliwości zastosowania sieci CP jest użycie jej w wersji autoasocjacyjnej do realizacji dowolnego odwzorowania

$$\mathbf{X} \Rightarrow \mathbf{Y}$$

przy czym ta sama sieć p nauczeniu może również realizować odwzorowanie

$$\mathbf{Y} \Rightarrow \mathbf{X}$$

co daje pole do ciekawych i wartościowych zastosowań.
Omawiana sieć ma strukturę pokazaną na rysunku:



Jak widać, na jej wejścia podaje się zarówno wektor $\mathbf{X}$, jak i wektor $\mathbf{Y}$ (zakladając, że liczba wejść sieci jest równa sumarycznej długości wektorów $\mathbf{X}$ i $\mathbf{Y}$, natomiast na wyjściach oczekuje się pojawienia się wektorów $\mathbf{X}'$ i $\mathbf{Y}'$ identycznych z wektorami wejściowymi $\mathbf{X}$ i $\mathbf{Y}$ (na tym polega autoasocjacyjność sieci).

Uczenie sieci przebiega wedlug następującego schematu. Kolejne elementy ciągu uczącego

$$U = \{< \mathbf{X}^{(1)}, \mathbf{Y}^{(1)} >, < \mathbf{X}^{(2)}, \mathbf{Y}^{(2)} >, \ldots, < \mathbf{X}^{(k)}, \mathbf{Y}^{(k)} >, \ldots, < \mathbf{X}^{(N)}, \mathbf{Y}^{(N)} >\}$$

podaje się równocześnie na oba wejścia sieci (jako sygnaly wejściowe) i na jej wyjścia (jako oczekiwane sygnaly wyjściowe). Sieć więc na tym etapie uczona jest realizacji odwzorowania tożsamościowego czyli wiernego odtwarzania na swoim wyjściu stanu swoich wejść. Po zakończeniu procesu nauki eksploatuje się natomiast sieć w sposób wielce oryginalny: podaje się na przyklad na wejście pewien sygnal $\mathbf{X}^{(k)}$, natomiast wejści $\mathbf{Y}$ pozostawione są bez sygnalów (odpowiada to sytuacji $\mathbf{Y} = \mathbf{0}$). Sieć na swoich wyjściach odtworzy sygnal $\mathbf{X}^{(k)}$, ale co więcej — na drugiej części wyjść odtworzy także sygnal $\mathbf{Y}^{(k)}$, który na etapie uczenia byl kojarzony z $\mathbf{X}^{(k)}$. Jak z tego wynika, sieć nauczyla się realizować odwzorowanie $\mathbf{X} \Rightarrow \mathbf{Y}$.

Ponieważ podczas uczenia sygnaly $\mathbf{X}$ i $\mathbf{Y}$ są calkowicie równoprawne, sieć potrafi także odtwarzać odwzorowanie odwrotne $\mathbf{Y} \Rightarrow \mathbf{X}$, wystarczy w tym celu na jej wejścia podać sygnal $\mathbf{Y}^{(k)}$ pozostawiając wejście $\mathbf{X}$ bez jakichkolwiek informacji, a na wyjściu otrzyma się sygnal $\mathbf{X}^{(k)}$ obok oczywiście odtworzonego przez sieć sygnalu $\mathbf{Y}^{(k)}$.

Sieci CP znalazly liczne zastosowania. Doskonale zdają one egzamin jako systemy klasyfikacji i rozpoznawania obrazów, są wykorzystywane w automatyce i robotyce do sterowania

określonych systemów, są wreszcie także bardzo cenione jako systemy służące do redukcji ilości przesylanych informacji — na przyklad podczas transmisji obrazów. Zwlaszcza to ostatnie zastosowanie, zwane kwantyzacją wektorową *(vector quantization)* ma obszerną literaturę i liczne udokumentowane zastosowania praktyczne.

Rozdział 6

Sieci rezonansowe

6.1 Podstawowy schemat działania sieci

Jedną z bardziej znanych zasad budowy sieci neuronowych jest ART *(Adaptive Resonance Theory)*. Teoria sieci ART oparta jest na pracach Stephana Grossberga, a konkretne metody wywodzące się z tej teorii są chronione patentem, którego posiadaczem jest Uniwersytet w Boston. Teoria ta dość silnie opiera się na biologicznych przeslankach — wykorzystano w niej znane z fizjologii zasady funkcjonowania rzeczywistych neuronów, zwlaszcza procesy przebiegające na blonie komórkowej. Prezentując tę teorię w książce dokonano w niej pewnych modyfikacji, wprowadzając oznaczenia i symbole graficzne zgodne z używanymi przy opisywaniu innych algorytmów. Oryginalne prace Grossberga i jego naśladowców wyróżniają się odmienną od ogólnie przyjętej notacją wzorów i symboliką rysunków; co utrudnia prowadzenie porównań pomiędzy metotodą ART, a innymi metodami uczenia sieci, dlatego zostalo to zmienione.

W literaturze wyróżnia się sieci ART 1 i ART 2. Sieć ART 1 służy do analizy obrazów binarnych, zaś ART 2 — analogowych. Każda z tych sieci sklada się z dwóch jednakowo licznych warstw neuronów — pierwszej (dolnej) rejestrującej sygnaly wejściowe (cechy rozpoznawanych obrazów) i drugiej, analizującej te cechy na zasadzie budowy ważonej sumy ich wejść i wypracowującej decyzje na zasadzie rywalizacji pomiędzy neuronami. Nie buduje się sieci ART o większej liczbie warstw, chociaż w teorii, sieć taka mogłaby wykazywać interesujące wlasności w zakresie przetwarzania informacji w systemach sensorycznych.

Oznaczmy sygnaly wyjściowe neuronów pierwszej (dolnej) warstwy przez y_i^d $(i = 1, 2, \ldots, n)$ oraz sygnaly neuronów drugiej warstwy przez y_i^g $(i = 1, 2, \ldots, k)$. Liczba neuronów n warstwy wejściowej (dolnej) uzależniona jest od liczby cech wyróżnionych w rozpoznawanych obiektach. W dalszych rozważaniach chętnie będziemy się odwolywali do przykladów, w których sygnalami docierającymi do warstwy wejściowej będą wartości informujące o jasności określonego obrazu (np. rysunku litery) w określonym punkcie. Źródlem sygnalów wejściowych będzie więc swoista siatkówka, binarny raster o wymiarach n_1 wierszy i n_2 kolumn. Przy takim założeniu oczywiście $n = n_1 n_2$. Liczba neuronów k warstwy wyjściowej jest w zasadzie dowolna, jednak ustala się ją zwykle mając na uwadze zadanie realizowane przez sieć. Ponieważ ART jest siecią klasyfikującą bodźce lub rozpoznającą obrazy, a ponadto jej uczenie zachodzi na zasadzie uczenia bez nauczyciela *(unsupervised learning)* przeto k musi być większe od liczby przewidywanych klas. Przykladowo rozpoznając duże litery

alfabetu lacińskiego (których jest 26) trzeba mieć około czterdziestu neuronów w warstwie wyjściowej, gdyż prawdopodobnie różniące się warianty tej samej litery zostaną zidentyfikowane przez kilka oddzielnych neuronów.

W górnej warstwie tylko jeden neuron ma sygnał wyjściowy, różny od zera

$$\exists!\, z,\ y_z^g > 0$$

Jest to ten, którego suma ważonych sygnałów wejściowych osiągnęła największą wartość

$$\forall\, j \neq z \left(\sum_{i=1}^{n} w_{ij}^g\, y_i^d < \sum_{i=1}^{n} w_{iz}^g\, y_i^d \right)$$

Jego sygnał wyjściowy przyjmowany jest jako równy 1, pozostałe neurony mają sygnały wyjściowe wyzerowane

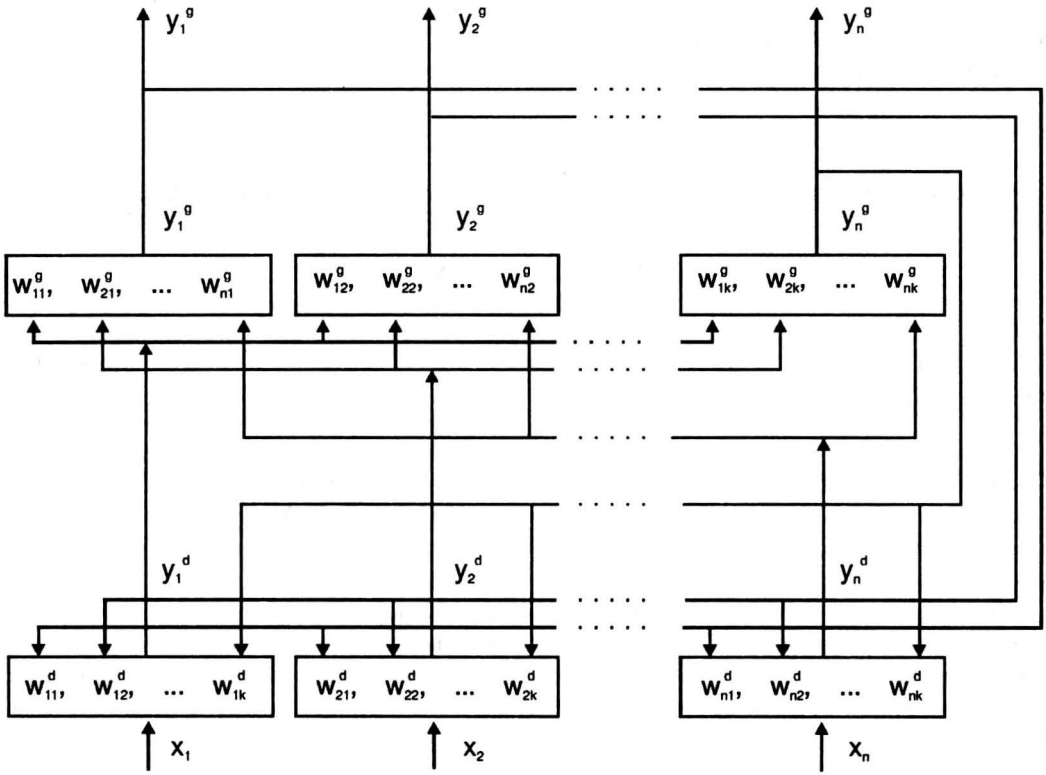
$$\forall\, j \neq z,\ y_j^g = 0$$

Taki schemat działania jest znany z wielu opracowań i (z dokładnością do kilku mało ważnych szczegółów) odpowiada omówionym wyżej koncepcjom sieci typu Madaline czy Perceptron. Element rywalizacji neuronów warstwy wyjściowej także występuje w koncepcjach wielu autorów, na przykład Kohonena. Na czym więc polega oryginalność sieci ART?

Dostrzeżoną przez Grossberga wadą sieci typu Madaline czy Perceptron jest **niestabilność ich zachowania w trakcie procesu uczenia**. Neuron warstwy wyjściowej, który w wyniku procesu uczenia uzyskał zdolność rozpoznawania pewnego obrazu — na przykład litery B — może w wyniku dalszego procesu uczenia zostać „przestawiony" na rozpoznawanie innego obrazu — na przykład litery R, co dość istotnie utrudnia korzystanie w takiej sieci (trudno być pewnym, jak się ona w konkretnej sytuacji zachowa). Dlatego w sieci ART wprowadzono dodatkowe połączenia biegnące od warstwy wyjściowej z powrotem do warstwy wejściowej. „Zwycięski" neuron warstwy wyjściowej może dzięki temu „wzmacniać" sygnały w tych neuronach warstwy wejściowej, od których otrzymywał pobudzenie. Przebadajmy ten proces bliżej.

Zdefiniujmy współczynniki wagowe przy odpowiednich połączeniach w sposób następujący. Niech w_{ij}^{ξ} będzie współczynnikiem wagowym połączenia między neuronami o numerach i oraz j, przy czym jeśli $\xi = g$, to odpowiednia waga znajduje się przy połączeniu wiodącym **do góry** czyli na drodze pomiędzy pierwszą warstwą, a drugą (we wstępującym nurcie przepływu sygnałów), natomiast jeśli $\xi = d$, to odpowiednia waga znajduje się przy połączniu wiodącym **w dół** czyli w torze sprzężenia zwrotnego — między drugą warstwą, a pierwszą. Warto podkreślić, że pierwszy indeks (i) oznacza zawsze numer neuronu w pierwszej warstwie, a drugi (j) w drugiej. Oczywiście i przebiega przedział od 1 do n a j — od 1 do k. Podkreślenie to jest o tyle potrzebne, że w sieci ART neurony są numerowane indywidualnie w każdej warstwie, zatem zapis — przykładowo — w_{ii}^{ξ} oznacza połączenie pomiędzy neuronami o tych samych numerach, a nie sprzężenie zwrotne zamykające się na tym samym neuronie.

Przytoczony poniżej na rysunku schemat sieci, jest jeszcze niekompletny. W toku dalszych rozważań wprowadzone zostaną dwa dalsze elementy, bardzo istotnie uzupełniające schemat tej sieci. Jednak dla wstępnych rozważań na temat działania i zachowania tej sieci celowe będzie rozpoczęcie dyskusji od tego uproszczonego modelu, który będziemy potem systemtycznie uzupełniali o kolejne dalsze elementy.



Jak wynika z przytoczonego opisu i z rysunku, neurony warstwy wejściowej są pod wpływem dwojakiego rodzaju sygnałów: biegnących od dołu do góry sygnałów wejściowych x_i oraz biegnących od góry do dołu sygnałów z warstwy wyjściowej y_i^g. Będziemy chwilowo zakładali, że wszystkie rozważane sygnały są binarne, tzn.

$$x_i \in \{0,1\} \quad , \quad y_i^g \in \{0,1\} \quad \text{i} \quad y_i^d \in \{0,1\}$$

co bardzo uprości rozważania. Te elementy, w których informacje dochodzące z obydwu kierunków nakładają się na siebie, uzyskują dodatkowe wzmocnienie. Dochodzi dzięki temu do swoistego **rezonansu** pomiędzy sygnałami zewnętrznymi i sygnałami wewnątrz sieci, co uzasadnia nazwę sieci.

6.2 Uczenie sieci ART

Proces uczenia sieci ART definiowany jest prawem Webera i może być opisany na kilka sposobów. W literaturze często podawany jest „uciąglony" wariant metody ART, w którym proces uczenia stanowi rozwiązanie układu równań różniczkowych. Taki sposób opisu (nawiązujący do równań dynamiki potencjałów czynnościowych występujących w rzeczywistym

neuronie) podali w kilku swoich oryginalnych pracach Gail Carpenter i Stephen Grossberg [Carp87], [Gros89]. W tej książce konsekwentnie stosowany jest opis polegający na prezentacji uczenia jako iteracyjnego procesu, w którym wartości wag w k-tym kroku procesu uczenia zmieniają się według pewnej reguły w zależności od wartości wag uprzednio zdefiniowanych oraz od sygnałów pojawiających się w sieci w k-tym kroku. Taki sposób pozwala na pokazanie istoty uczenia w formie uproszczonej, chociaż nie pozbawionej praktycznej przydatności (każdy program symulujący sieć będzie zawsze obliczał zmiany wag w formie iteracji, a nie w formie procesu ciągłego, opisanego równaniem różniczkowym). W tym uproszczonym, prezentowanym tu modelu wagi w_{ij}^{d} zmieniają się w sposób następujący:

$$w_{ij}^{d(k+1)} = \begin{cases} 1 & \text{gdy} \quad (y_i^{d(k)} = 1) \wedge (y_j^{g(k)} = 1) \\ w_{ij}^{d(k)} & \text{gdy} \quad y_j^{g(k)} = 0 \\ 0 & \text{gdy} \quad (y_i^{d(k)} = 0) \wedge (y_j^{g(k)} = 1) \end{cases}$$

gdzie indeks k pojawiający się przy poszczególnych w i y jest numerem kroku w procesie uczenia.

Warto zauważyć, że podana reguła uczenia nie jest regułą typu Hebba ze względu na czynnik zapominania, wyrażony ostatnim wierszem przytoczonego wzoru. Warto także zauważyć, że z powodu rywalizacji w warstwie wyjściowej w rzeczywistości jedynie nieliczne współczynniki wagowe podlegają uczeniu, gdyż dla wszystkich j z wyjątkiem $j = z$ zachodzi $y_j^g = 0$. **Wartości początkowe** nadawane przed procesem uczenia połączeniom „z góry na dół" są jednakowe i wynoszą $w_{ij}^{d(1)} = 1$. Warto zauważyć, że w tym przypadku wartości początkowe **nie są losowe**.

Reguła uczenie wag połączeń wiodących w górę jest podobna do wyżej opisanej, ale bardziej subtelna (to znaczy zmiany wartości wag nie są tak gwałtowne i radykalne). Regułę tę można zapisać w następującej postaci:

$$w_{ij}^{g(k+1)} = \begin{cases} w_{ij}^{g(k)} + \delta_1 & \text{gdy} \quad (y_i^{d(k)} = 1) \wedge (y_j^{g(k)} = 1) \\ w_{ij}^{g(k)} & \text{gdy} \quad y_j^{g(k)} = 0 \\ w_{ij}^{g(k)} - \delta_2 & \text{gdy} \quad (y_i^{d(k)} = 0) \wedge (y_j^{g(k)} = 1) \end{cases}$$

Przyrosty δ_1 i δ_2 nadawane współczynikom wagowym w trakcie procesu uczenia mogą być przyjmowane jako stałe, ale lepsze wyniki uczenia i dokładniejsze dopasowanie do rozpoznawanych klas uzyskuje się przyjmując te wartości jako zależne od liczby pobudzonych neuronów w pierwszej warstwie. Liczbę tę można wyznaczyć jako

$$p^{(k)} = \sum_{i=1}^{n} y_i^{d(k)}$$

ponieważ wszystkie $y_i^{d(k)}$ są binarne. Mając wyznaczoną liczbę $p^{(k)}$ można ustalać δ_1 i δ_2 z dokładnych wzorów wyprowadzonych przez Carpentera

$$\begin{aligned} \delta_1 &= \eta\,(1 - w_{ij}^{g(k)}) - w_{ij}^{g(k)}\,(p^{(k)} - 1) \\ \delta_2 &= w_{ij}^{g(k)}\,p^{(k)} \end{aligned}$$

lub z prostszych formul wypraktykowanych m.in. przez firmę NeuralWare

$$\delta_1 = \frac{\eta}{\eta - 1 + p^{(k)}} - w_{ij}^{g(k)}$$
$$\delta_2 = w_{ij}^{g(k)}$$

Parametr η występujący w powyższych wzorach musi być dobierany empirycznie; z teorii wynika, że $\eta > 1$ i większe wartości tego współczynnika sprzyjają tworzeniu przez sieć nowych wzorców kategorii, mniejsze natomiast skłaniają ją do uogólniania kategorii już wcześniej wypracowanych.

Wartości początkowe przyjmowane dla wag $w_{ij}^{g(1)}$ przyjmowane są losowo z przedziału $[0, 1/n]$ (tym mniejsze, im więcej elementów liczy warstwa wejściowa).

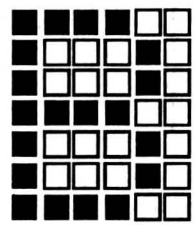
W wyniku takiej radykalnej i wręcz brutalnej adaptacji wag (patrz niżej) możliwe jest dopasowywanie działania sieci do wielu wzorców różniących się od siebie, a ponadto sieć wykazuje pewną (ograniczoną) zdolność do uśredniania sygnałów wejściowych i tworzenia uogólnionych wzorców rozpoznawanych obiektów. Ta właściwość sieci jest podobna do analogicznych cech wykazywanych przez sieć Madaline, Perceptron lub Backpropagation i nie wymaga tu szerszej dyskusji.

Opisana wyżej reguła uczenia sieci ART nie jest jedynym możliwym algorytmem jej adaptacji do zmiennych zadań. Jednak ta reguła jest na tyle prosta i czytelna w kontekście wcześniejszych rozważań, że możliwe jest jej wykorzystanie w celu przemyślenia zasad działania sieci ART, jej zalet i wad.

6.3 Zasada działania sieci ART

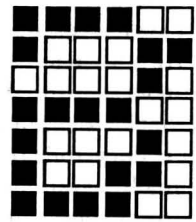
Rozważmy przykład działania sieci rozpoznającej litery. Niech wejściowy sygnał podawany do neuronów pierwszej warstwy ma postać:

X =

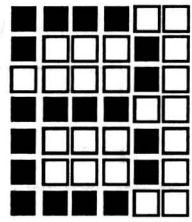


Zaciemnione elementy ■ podanego rastra pokazują elementy przesyłające sygnał wejściowy o wartości $x_i = 1$, a puste elementy □ odpowiadają wartościom $x_i = 0$.

W wyniku tego pobudzenia pewnien neuron wyjściowej warstwy został pobudzony ($y_z^g = 1$), co można uznać za sukces (poprawne rozpoznanie). Można założyć *(ex post)*, że był to neuron o numerze odpowiadającym rozpoznaniu litery B. Ponieważ sieć funkcjonuje bez nauczyciela, nie było możliwe wskazanie tego neuronu *a priori*, przed jego ujawnieniem w toku uczenia, ale po pierwszym skojarzeniu konkretnego neuronu wyjściowego z konkretnym wejściowym sygnałem sieć ART utrzymuje to powiązanie we wszystkich dalszych pokazach w sposób stabilny i zdeterminowany. Jednak rzutowanie wstecz $\mathbf{Y}^g$ do wejściowej warstwy obszaru, z którego pochodziły pobudzenia wyróżnionego neuronu wykazuje zwykle, że obszar ten nie jest całkowicie poprawny. Przykładowo może on wyglądać tak:

$\mathbf{Y}^g =$ 

Wówczas obszar, w którym spotykają się sygnały wyjściowe $\mathbf{Y}^g$ z wejściowymi $\mathbf{X}$ ulega wzmocnieniu na zasadzie teorii rezonansu. Obszar ten, oznaczany czasem $\mathbf{Y}^g \cap \mathbf{X}$ ma w omawianym przykładzie postać

$\mathbf{Y}^g \cap \mathbf{X} =$ 

i jest lepiej dopasowany do realizacji postawionego przed siecią zadania klasyfikacji od pierwotnego obszaru pobudzeń „zwycięskiego" elementu wyjściowej warstwy sieci. W pierwszym przybliżeniu można przyjąć, że obszar $\mathbf{Y}^g \cap \mathbf{X}$ odpowiada obszarowi, w którym sygnały pierwszej warstwy są niezerowe, a więc rozkład pobudzeń pierwszej warstwy (pomijając pewne bardziej subtelne mechanizmy) można zapisać jako

$$\boldsymbol{Y}^d = \boldsymbol{Y}^g \cap \boldsymbol{X}$$

Działanie mechanizmu uczenia jest — jak wynikało z przytoczonych uprzednio wzorów — dość złożone i subtelne, ale **w przybliżeniu** może być ono opisane następująco. Modyfikacji podlegają jedynie wagi połączeń w_{iz}^g dochodzących do „zwycięskiego" elementu warstwy wyjściowej oraz wagi połączeń w_{iz}^d $(i = 1, 2, \ldots, n)$ biorących początek w „zwycięskim" neuronie. Wszystkie pozostałe połączenia pozostają nie zmienione.

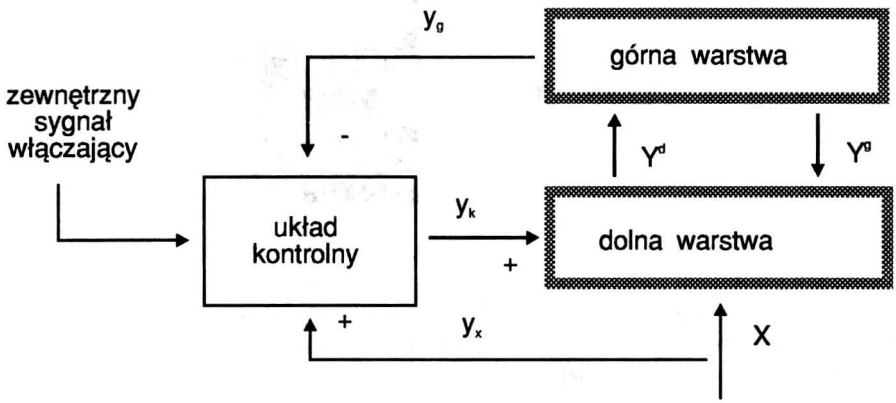
Modyfikacja połączeń polega (z grubsza) na tym, że wszystkie wagi w_{iz}^d odpowiadające połączeniom z pobudzonymi elementami pierwszej warstwy ($\mathbf{Y}^d = \mathbf{Y}^g \cap \mathbf{X}$) przymują wartości 1, a wszystkie pozostałe 0. Warto zauważyć, że jest to zabieg dość brutalny i pozbawiony subtelności charakterystycznych dla niektórych innych metod uczenia! Podobnie modyfikowane są wagi połączeń „w górę" w_{iz}^g — ich wartości są zerowane dla wszystkich niepobudzonych elementów warstwy wejściowej, względnie są ustawiane na pewną wartość zależną od liczby pobudzonych elementów w przypadku połączeń biorących początek w tych pobudzonych neuronach.

6.4 Rola i struktura układu kontrolnego

Opisane wyżej działanie sieci ART dotyczyło sytuacji, kiedy na wejście sieci podawane są określone, niezerowe sygnały $\mathbf{X}$. W takim wypadku działanie sieci jest — jak wykazano wyżej — sensowne i celowe. Jeśli jednak na wejściu sieci nie ma sygnałów ($\mathbf{X} = 0$), to może się zdarzyć, że na którymś z wyjść drugiej warstwy pojawi się na chwilę sygnał $y_j^g \neq 0$. Na

skutek sprzężenia zwrotnego sygnał $\mathbf{Y}^g$ dotrze do wszystkich neuronów pierwszej warstwy, wywołując ich aktywizację. Aktywizacja neuronów pierwszej warstwy $\mathbf{Y}^d$ spowoduje pobudzenie neuronów warstwy drugiej $\mathbf{Y}^g$ itd. W sieci pojawi się złożony proces dynamiczny, nie mający literalnie żadnego związku z syganalami docierającymi do sieci z zewnątrz i z jej podstawowymi zadaniami, a będący rodzajem symulowanej halucynacji.

Oczywiście w sieci, która ma być systemem przetwarzającym informacje w sposób powtarzalny i ukierunkowany na określony cel, takie zjawiska są niedopuszczalne. Dlatego elementem sieci ART jest zawsze układ kontrolny *(attentional gain control unit)*, który uniemożliwia pierwszej warstwie sieci reagowanie na sygnały sprzężenia zwrotnego w przypadku braku sygnałów wejściowych. Schemat włączenia tego elementu prezentuje rysunek.



Działanie układu kontrolnego polega na tym, że jego sygnał dodatkowo pobudza (albo zwiększa czułość) neuronów dolnej warstwy. Bez tego dodatkowego sygnału neurony te nie są w stanie reagować na sygnały z górnej warstwy $\mathbf{Y}^g$ i pętla „halucynacji" ulega przerwaniu. Układ kontrolny jest odblokowany i dostarcza swój sygnał w przypadku pojawienia się w sieci sygnału wejściowego $\mathbf{X}$ (symbol „+" na rysunku), natomiast może być blokowany przez sygnały pochodzące z górnej warstwy („-") oraz przez zewnętrzny sygnał wyłączający, który ma interpretację konkurencji ze strony innych systemów informacyjnych. W pracach na temat sieci ART autorzy chętnie operują tu porównaniem z sytuacją uczestnika przyjęcia, któremu interesująca rozmowa uniemożliwia dokładne rozpoznanie smaku wina. Jak widać sieć ART ma służyć jako model rzeczywistego zachowania człowieka, a nie tylko jako narzędzie używane w celu sprawniejszego przetwarzania informacji — co stanowi o pewnej wyjątkowości tej sieci.

Realizacja układu kontrolnego zakłada zwykle dobudowanie do sieci dodatkowego układu złożonego z trzech neuronów. Pierwszy z tych neuronów nazywa się detektorem wejścia i bada stan wszystkich wejść $\mathbf{X}$. Jego działanie można opisać jako *alternatywę* sygnałów wejściowych, a formuła, według której działa może być opisana wzorem

$$y_x = \begin{cases} 1 & \text{gdy} \quad \sum_{i=1}^{n} x_i > 0 \\ 0 & \text{gdy} \quad \sum_{i=1}^{n} x_i \leq 0 \end{cases}$$

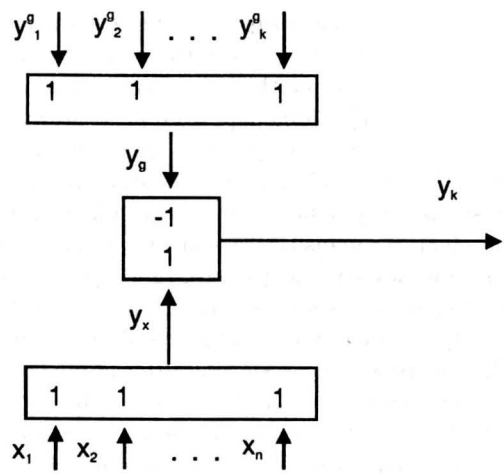
W podobny sposób działa drugi neuron, badający aktywność $\mathbf{Y}^g$ drugiej warstwy neuronów

$$y_g = \begin{cases} 1 & \text{gdy} \quad \sum_{i=1}^{k} y_i^g > 0 \\ 0 & \text{gdy} \quad \sum_{i=1}^{k} y_i^g \leq 0 \end{cases}$$

Progową nieliniową charakterystykę ma też trzeci neuron tworzący sygnał wyjściowy y_k układu kontrolnego, z tym jednak, że ma on zaledwie dwa wejścia:

$$y_k = \begin{cases} 1 & \text{gdy} \quad y_x - y_g > 0 \\ 0 & \text{gdy} \quad y_x - y_g \leq 0 \end{cases}$$

Strukturę układu kontrolnego przedstawić więc można następująco:



Sygnał y_k doprowadzony jest do wszystkich elementów dolnej (wejściowej) warstwy sieci i współuczestniczy w kształtowaniu ich sygnałów wyjściowych zgodnie z równaniem

$$y_i^d = \begin{cases} 1 & \text{gdy} \quad x_i + \sum_{j=1}^{k} w_{ij}^d \, y_j^g + y_k > \Theta \\ 0 & \text{gdy} \quad x_i + \sum_{j=1}^{k} w_{ij}^d \, y_j^g + y_k \leq \Theta \end{cases}$$

Równanie to odwzorowuje strukturę połączeń elementów sieci, jednak z punktu widzenia aktywnych sygnałów występujących w niej podane wyżej równanie jest całkowicie równoważne prostszej zależności

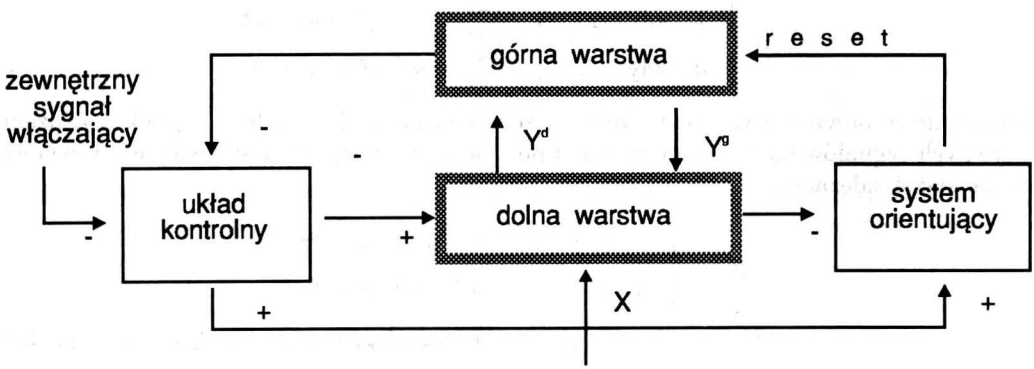
$$y_i^d = \begin{cases} 1 & \text{gdy} \quad x_i + w_{iz}^d \, y_z^g + y_k > \Theta \\ 0 & \text{gdy} \quad x_i + w_{iz}^d \, y_z^g + y_k \leq \Theta \end{cases}$$

w której zapisie uwzględniono fakt, że $y_j^g = 0$ dla wszystkich $j \neq z$, co znacznie uprościło zapis wzoru. Próg Θ przyjmuje się taki, by spełniona była reguła „dwa z trzech" (tzn. do zadziałania neuronu potrzebna jest obecność dwóch spośród trzech jego sygnałów wejściowych). Zwykle przyjmuje się $\Theta = 1,5$.

6.5 Układ orientujący

Innym oryginalnym elementem sieci ART jest system orientujący *(orienting system)*, którego celem jest sterowanie precyzją odwzorowania poszczególnych kategorii w sieci ART. System ten określany bywa także jako detektor nowych danych *(novelity detector)* i to określenie szczególnie wyraźnie określa jego rolę. Strukturę połączeń systemu orientującego z elementami sieci, które były uprzednio omówione, przedstawia rysunek. Widać na nim, że system orientujący jest pobudzany przez sygnał wejściowy i hamowany przez sygnał globalnej aktywności dolnej warstwy sieci. System ten pozostaje nieaktywny w przypadku, kiedy sygnałowi wejściowemu $\mathbf{X}$ towarzyszy reakcja w postaci pobudzenia dolnej warstwy sieci $\mathbf{Y}^d$, gdyż wtedy sieć po prostu rozpoznaje jeden z wcześniej zapamiętanych wzorców wejściowego sygnału i żadne dodatkowe działania nie są potrzebne. Załóżmy jednak, że na wejściu pojawił się nowy bodziec $\mathbf{X}$. Sieć nie zna go, zatem nie dochodzi do „rezonansu" i pobudzenie dolnej warstwy jest niewielkie. Jednak jest to przedstawiciel nowej kategorii sygnałów, które także powinny być w sieci zapamiętane, zatem trzeba spowodować jego rejestrację w postaci odpowiedniej reprezentacji w górnej warstwie, przy czym powinna to być reprezentacja **inna**, niż w przypadku wszystkich wcześniej zapamiętanych bodźców.

Istotnie, połączenia warstw górnej i dolnej spowodują, że nowy bodziec wejściowy pobudzi pewne neurony w górnej warstwie, a te poprzez sprzężenie zwrotne będą oddziaływały na neurony warstwy dolnej. Jednak brak korelacji między uprzednio zapamiętanymi wzorcami sygnałów, a nowym sygnałem jest powodem, że wynikowe pobudzenie warstwy dolnej zmaleje — nie pojawi się efekt rezonansu. Ta właśnie sytuacja aktywizuje system orientujący. Wysyła on do neuronów warstwy wyjściowej (górnej) krótkotrwały sygnał blokujący *(reset wave)*, który powoduje, że wszystkie aktywne neurony warstwy górnej zostają zablokowane. Wyłącza to w praktyce oddziaływanie sygnału $\mathbf{Y}^g$ na dolną warstwę i powoduje, że rozkład pobudzeń na dolnej warstwie może się kształtować w sposób swobodny, bez wprowadzających deformacji oddziaływań zwrotnych. Prowadzi to do wzrostu poziomu sygnału $\mathbf{Y}^d$ i silnego pobudzenia tych elementów warstwy górnej, które nie podlegały blokadzie; jeden z tych syganlów staje się „zwycięzcą" i neuron odpowiadający temu sygnałowi staje się detektorem nowej klasy. Następuje stosowna korekta wag połączeń synaptycznych i sieć nabiera zdolności rozpoznawania nowej klasy; zdolność ta jest następnie doskonalona w toku dalszych pokazów.



Opisany mechanizm ma dwie istotne zalety. Z jednej strony pozwala on na wprowadzanie nowych klas do rozpoznawanego zbioru bez interferencji ze strony klas już wcześniej podlegających rozpoznawaniu. Z drugiej zapobiega zjawisku przepełnienia pamięci, gdyż w momencie, kiedy wszystkie neurony wyjściowej warstwy mają już swoje „przypisane" wzorce sygnałów wejściowych, następuje ich całkowita blokada i nowe wzorce, pojawiające się w trakcie procesu uczenia, nie powodują efektu „przeuczania" sieci z zapominaniem poprzednich wzorców.

System orientujący pełni w sieci ART rolę „zastępczego nauczyciela". Istotnie, uczenie w tej sieci ma charakter uczenia bez nauczyciela, gdyż w ciągu uczącym brak jawnie zadawanych wzorców sygnałów wyjściowych z sieci. Jednak zachowanie sieci nie jest całkiem przypadkowe, gdyż system orientujący klasyfikuje pojawiające się wzorce wejściowych sygnałów na „znane" i „nowe", dzięki czemu poszczególne neurony warstwy wyjściowej nabywają zdolności do rozpoznawania i sygnalizowania różnych wzorców.

Działanie systemu orientującego zależne jest od parametru gotowości v *(vigilance)* określającego, jak duża musi być niezgodność między sygnałem wejściowym $\mathbf{X}$, a sygnałem zwrotnym $\mathbf{Y}^g$ aby można było mówić o braku rezonansu i aby dochodziło do aktywacji sygnału blokowania *(reset)*. Jeśli parametr v jest duży — tylko małe niezgodności pomiędzy sygnałem wejściowym, a jego wewnętrzną reprezentacją są tolerowane i łatwo dochodzi do generacji sygnału blokującego i wykrycia nowej kategorii. Małe wartości v czynią sieć bardziej skłonną do uśredniania wejściowych sygnałów i generacji bardziej „zagregowanych" reprezentacji. Duże wartości v odpowiadają więc sytuacji, kiedy potrzebujemy sieci zdolnej do generowania dużej liczby subtelnych rozróżnień wśród wejściowych sygnałów podawanych do sieci, natomiast małe wartości v są polecane dla sieci, która ma dokonywać jedynie bardzo ogólnych klasyfikacji. Wybór konkretnej wartości jest zawsze kwestią swobodnej decyzji użytkownika sieci.

Formalny opis systemu orientującego jest bardzo prosty. Wyznaczany jest iloraz sumy odpowiedzi neuronów pierwszej warstwy $\sum_{i=1}^{n} y_i^d$ do sumy zewnętrznych sygnałów wejściowych $\sum_{i=1}^{n} x_i$. Jeśli iloraz ten jest mniejszy od zadanej wartości parametru v, wówczas generowany jest sygnał blokujący r *(reset wave)*. Można to zapisać w formie warunku

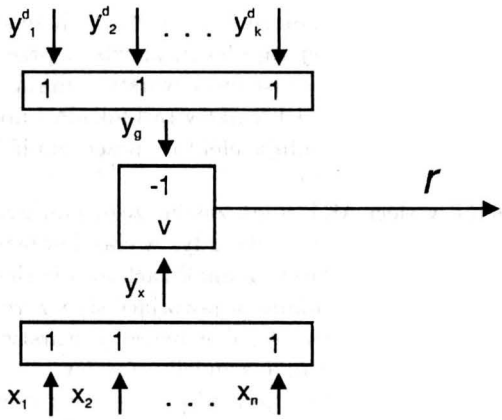
$$v \sum_{i=1}^{n} x_i > \sum_{i=1}^{n} y_i^d$$

i opisać działanie sytemu orientującego łącznym wzorem

$$r = \begin{cases} 1 & \text{gdy} \quad v \sum_{i=1}^{n} x_i > \sum_{i=1}^{n} y_i^d \\ 0 & \text{gdy} \quad v \sum_{i=1}^{n} x_i \leq \sum_{i=1}^{n} y_i^d \end{cases}$$

System ten daje się zrealizować za pomocą układu trzech neuronów o budowie zbliżonej do budowy wcześniej omówionego układu kontrolnego, (rysunek poniżej).

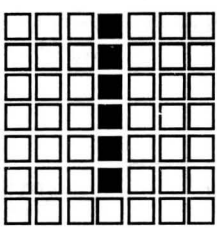
Warto jednak zauważyć, że podobnie wyglądające fragmenty sieci systemu orientującego i układu kontrolnego pełnią jednak odmienną funkcję — w omawianym tu systemie sygnały sumy wejść i sumy wyjść pierwszej warstwy neuronów traktować trzeba jako sygnały analogowe, przyjmujące dowolne wartości z przedziału $[0, n]$, podczas gdy sygnały y_x i y_g występujące w strukturze układu kontrolnego były cyfrowe (przyjmowały wyłącznie wartości 0 i 1).

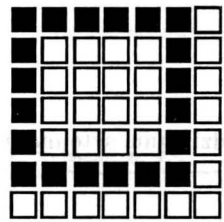


Warto także odnotować konieczność empirycznego dobierania wagi synaptycznej oznaczonej jako v — zgodnie z przytoczonymi rozważaniami.

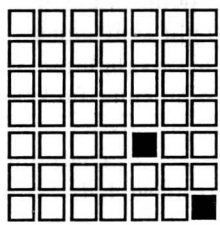
6.6 Mankamenty sieci ART

W istocie system orientujący posługuje się swoistym „modelem" zadania rozwiązywanego przez sieć. Model ten zakłada, że sygnały należące do tej samej klasy mają większość cech wejściowych (składowych wektora **X**) identycznych, natomiast duża liczba różniących się cech wejściowych jest charakterystycznym wyróżnikiem sytuacji, kiedy pojawia się obiekt z nowej, nieznanej klasy. To założenie może być prawdziwe, ale może i nie być. Na przykład rozważmy sieć, która już nauczyła się rozpoznawać wzorce liter **I** oraz **O**. Sygnały wejściowe **X** dla tych rozpoznawanych klas mają postać

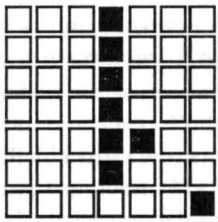


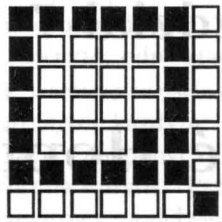


Wyobraźmy sobie teraz, że rozpatrujemy te same obrazy, ale zakłócone dodaniem dwóch pikseli według następującego wzoru



Trudno nie przyznać, że zakłócenie ma charakter drobny i nie powinno — w teorii — wpływać na kategoryzację rozważanych obrazów. Jednak nałożenie tych zakłóceń na pierwotne obrazy ujawnia następujący fakt. Zakłócona litera **I** nie przestaje być literą **I** (tu nasze oczekiwania odpowiadają rzeczywistości), natomiast zakłócone **O** okazuje się być innym znakiem, mianowicie **Q**





Sieć ART jest wobec takiej sytuacji kompletnie bezradna. Żadne manewrowanie współczynnikiem v nie zapewni sensownego działania sieci, gdyż udział „szumowych" pikseli w obrazie litery **I** jest znacznie większy, niż ich udział w obrazie litery **O**, a tymczasem właśnie w tym drugim przypadku powinno się zbudować nową klasę, a w pierwszym przypadku należy uznać, że mamy do czynienia ze znanym uprzednio bodźcem — tylko obecnie prezentowanym w formie zakłóconej.

Oczywiście w podanym przykładzie bardzo łatwo sformułować wniosek, że winna jest niewłaściwa reprezentacja obrazu. Istotnie, opierając się wyłącznie na obecności lub braku pikseli w konkretnych punktach siatkówki nie da się sensownie rozpoznawać znaków alfanumerycznych (potrzebne są tu bardziej wyrafinowane **cechy** obrazów rozpoznawanych liter), jednak nawet wprowadzenie takich doskonalszych cech nie uwolni całkowicie nas od częstych w praktyce sytuacji, kiedy drobna zmiana kilku z tych cech raz może oznaczać zasadniczą zmianę rozpoznania obrazu, a w innym wypadku duża zmiana cech może wiązać się jedynie z mało istotnym zakłóceniem. Dyskutowany wyżej przykład z siatkówką, na którą rzutowane są obrazy liter, stanowił jedynie łatwą do prześledzenia ilustrację istoty tych problemów.

Niezdolność sieci ART do rozróżniania drobnych, ale istotnych zmian rozpoznawanego obrazu od ewidentnych zakłóceń jest jej oczywistą wadą. Trzeba jednak stwierdzić, że nie jest to wada wyłącznie tej właśnie sieci; w istocie takie problemy w mniej lub bardziej istotny sposób pojawiają się przy użyciu dowolnej sieci wykorzystującej uczenie bez nauczyciela. Z tego względu w zastosowaniach mających rozwiązywać zagadnienia praktyczne preferuje się sieci realizujące zasadę *supervised learning*, na przykład z wsteczną propagacją błędów.

Rozdział 7

Sieć Hopfielda

7.1 Sprzężenie zwrotne jako nowa jakość w strukturach sieci

Dotychczas opisywane sieci charakteryzowały się jednokierunkowym przepływem sygnałów. Można w nich było wyróżnić warstwę wejściową, wyjściową i ewentualnie warstwy pośrednie („ukryte"). Jednak przepływ sygnałów w tych sieciach był ściśle jednokierunkowy: od wejścia do wyjścia. Najbardziej znaną siecią, w której kierunek ten jest odwrócony, jest sieć **Hopfielda**. Została ona zdefiniowana w historycznej pracy tego badacza, której publikacja w 1982 roku stała się punktem zwrotnym w badaniach sieci neuronowych i przywróciła zainteresowanie tymi systemami sferze dociekań naukowych. W sieci tej neurony mają nieliniowe charakterystyki:

$$y_m^{(j)} = \varphi\left(e_m^{(j)}\right)$$

gdzie

$$e_m^{(j)} = \sum_{i \in \mathfrak{M}} w_i^{(m)}\, y_i^{(j)} + x_m^{(j)}$$

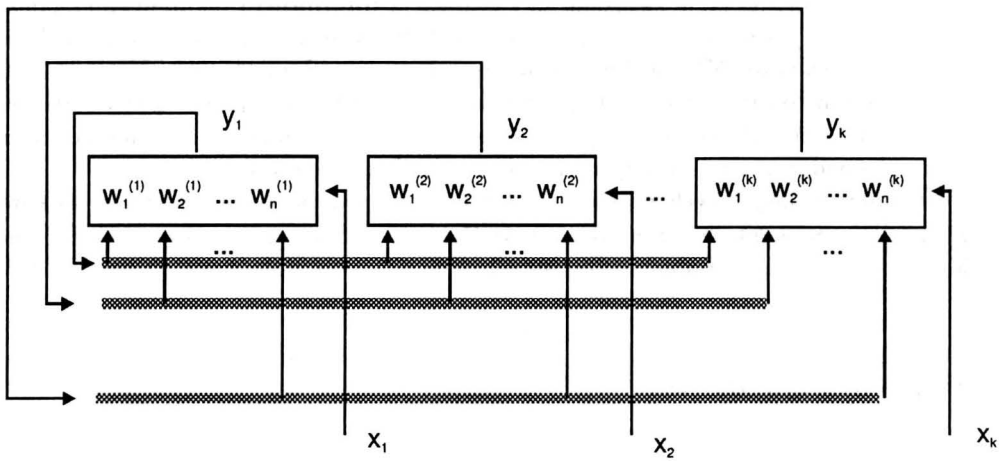
a nieliniowość $y = \varphi(e)$ dana jest prostą binarną funkcją

$$y_m^{(j+1)} = \begin{cases} 1 & \text{gdy} \quad e_m^{(j)} > w_0^{(m)} \\ y_m^{(j)} & \text{gdy} \quad e_m^{(j)} = w_0^{(m)} \\ -1 & \text{gdy} \quad e_m^{(j)} < w_0^{(m)} \end{cases}$$

Warto przy tym zwrócić uwagę na dwa szczegóły podanych wyżej wzorów. Otóż po pierwsze, współczynniki wagowe $w_i^{(m)}$ łączące wyjście i-tego neuronu z wejściem m-tego neuronu **nie zależą od** j. Wynika to z faktu, że rozważając sieć Hopfielda **na tym etapie** nie dyskutujemy problemu jej uczenia. Zakładamy, że wartości $w_i^{(m)}$ zostały wcześniej ustalone za pomocą jakiegoś algorytmu (najczęściej przyjmuje się tu algorytm Hebba) i obecnie nie podlegają zmianom. Numer j oznacza natomiast chwilę czasową, określającą w jakim momencie **procesu dynamicznego** następującego po pobudzeniu sieci obecnie się znajdujemy. Zagadnienie to będzie niżej dokładniej dyskutowane. Po drugie sumowanie sygnałów

wyjściowych $y_i^{(j)}$ z poszczególnych neuronów we wzorze definiującym łączne pobudzenie $e_m^{(j)}$ odbywa się po wszystkich elementach $i \in \mathfrak{M}$ czyli po **wszystkich** elementach sieci. Oznacza to, że w sieci przewidziane są także połączenia z warstw dalej położonych (wyjściowych) do warstw wcześniejszych — czyli **sprzężenia zwrotne**.

Jak bylo to wcześniej odnotowane, sieć o takim schemacie połączeń nazywana jest siecią **autoasocjacyjną**. W ramach tego sprzężenia każdy neuron jest także połączony jednym z wejść ze swoim własnym wyjściem, zatem zasada autoasocjacyjności odnosi się także do pojedynczych neuronów. **Każdy** neuron sieci ma także kontakt z pewnym, odpowiadającym mu sygnalem wejściowym $x_m^{(j)}$, zatem zaciera się tu podzial na warstwę wejściową i pozostale warstwy sieci. Podobnie w związku z zasadą laczenia „każdego z każdym" neurony sieci Hopfielda **nie tworzą wcale wyraźnie wydzielonych warstw** i mogą być rozważane w dowolnej topologii. My utrzymamy jednak nadal rysunki w formie warstw. Przyjmując w dalszych rozważaniach najprostszy, jednowarstwowy i jednowymiarowy model sieci otrzymujemy następujący jej schemat:



7.2 Natura procesów w sieciach Hopfielda

Warto zauważyć, że w sieciach autoasocjacyjnych możliwe jest pojawianie się pewnych przebiegów dynamicznych, polegających na tym, że uzyskane w pewnym kroku j wartości sygnalów wyjściowych $y_m^{(j)}$ wszystkich neuronów sieci ($m = 1, 2, \ldots, k$) stają się automatycznie wartościami wejściowymi $y_i^{(j+1)} (i \in \mathfrak{M}$ w kolejnym kroku symulacji. Oznacza to, że sieć realizuje **nieliniowe** wektorowe odwzorowanie

$$\boldsymbol{Y}^{(j+1)} = \boldsymbol{\Xi}\left(\boldsymbol{X}^{(j)}, \boldsymbol{Y}^{(j)}\right)$$

które zwykle ulega dodatkowemu uproszczeniu, ponieważ przyjmuje się, że $x_m^{(j)} \equiv 0$ dla wszystkich m i dla wszystkich $j > 0$. Uproszczenie to ma następującą interpretację. W chwili początkowej ($j = 0$) do neuronów sieci (wszystkich, albo tylko niektórych, wybranych) doprowadza się sygnały wejściowe $x_m^{(0)} \neq 0$. W wyniku tego na wyjściach neuronów sieci wytwarza się zestaw sygnałów wyjściowych $\mathbf{Y}^{(1)}$. W tym momencie sygnały wejściowe zostają odłączone i aż do końca symulacji nie uczestniczą w obliczeniach ($x_m^{(j)} \equiv 0$), natomiast w sieci zaczyna się rozwijać pewien proces, polegający na wyznaczaniu kolejnych wartości $\mathbf{Y}^{(j)}$ ($j = 2, 3, \ldots$) na podstawie zależności

$$\mathbf{Y}^{(j+1)} = \boldsymbol{\Xi}\,(\mathbf{Y}^{(j)})$$

Proces wyznaczany przez kolejne wartości

$$\mathbf{Y}^{(1)}, \mathbf{Y}^{(2)}, \mathbf{Y}^{(3)}, \ldots, \mathbf{Y}^{(j-1)}, \mathbf{Y}^{(j)}, \mathbf{Y}^{(j+1)}, \ldots$$

można obserwować w **przestrzeni stanu** $\mathcal{Y} \subseteq \mathcal{R}^k$, do której należą wszystkie wektory sygnałów wyjściowych z elementów sieci $\mathbf{Y}^{(j)}$. W tej przestrzeni możliwe są wszystkie szeroko znane procesy, jakie związane są z realizacją **nieliniowej** rekurencyjnej zależności $\mathbf{Y}^{(j+1)} = \Xi\,(\mathbf{Y}^{(j)})$, a mianowicie możliwe jest **stabilizowanie** się przebiegów i ich zbieżność do określonych wartości $\mathbf{Y}^*$, możliwe jest pojawianie się **oscylacji** wartości $\mathbf{Y}^{(j)}$ i związanych z nimi **cykli** oraz **orbit** w przestrzeni $\mathcal{Y}$, możliwe jest powstawanie przebiegów **rozbieżnych**, w ramach których wartości niektórych (lub wszystkich) $y_i^{(j)}$ zmierzają do nieskończoności, wreszcie przewidywać można w tym systemie pojawianie się **chaosu**.

O wyborze jednej z tych możliwości decyduje oczywiście zestaw współczynników wagowych $w_i^{(m)}$. Stosunkowo wcześnie (1983) Cohen i Grossberg wykazali, że sieć generuje stabilne rozwiązania, jeśli uniemożliwi się autoasocjacyjność pojedynczych neuronów

$$w_m^{(m)} = 0$$

oraz zapewni się symetrię sieci

$$w_i^{(m)} = w_m^{(i)}$$

to sieć wykazuje stabilność zachowania. Nawet jednak przy stabilnym zachowaniu sieci pozostaje otwarty problem wyboru przez sieć konkretnego stabilnego stanu docelowego $\mathbf{Y}^*$. Takich możliwych stanów jest oczywiście nieskończenie wiele i dlatego potrzebne jest precyzyjne kryterium, określające, który z nich zostanie przez sieć „wybrany".

7.3 Stany równowagi w sieci Hopfielda

Problem wyboru określonego „docelowego" stanu sieci można traktować jako problem wyboru stanu o mimimalnej „energii" sieci. Funkcja „energii" jest tu oczywiście wprowadzona w sposób czysto umowny, w rzeczywistości należy raczej mówić o **funkcji Lapunowa**, jednak większość autorów przyjmuje za Hopfieldem tę energetyczną metaforę. Funkcja „energii" może być dla sieci zdefiniowana w sposób następujący:

$$E^{(j)} = (-1/2) \sum_{i \in \mathfrak{M}} \sum_{m \in \mathfrak{M}} w_i^{(m)}\, y_i^{(j)}\, y_m^{(j)} - \sum_{i \in \mathfrak{M}} x_i^{(j)}\, y_i^{(j)} + \sum_{i \in \mathfrak{M}} w_0^{(i)}\, y_i^{(j)}$$

Na podstawie wyżej podanej definicji funkcji E można obliczyć **zmianę** δE zachodzącą na skutek zmiany stanu sieci wyrażającej się zmianą sygnalu wyjściowego i-tego neuronu:

$$\delta E^{(j)} = -\Big[\sum_{m \neq i} w_m^{(i)}\, y_m^{(j)} + x_i^{(j)} - w_0^{(i)}\Big]\, \delta y_i^{(j)}$$

Wzór ten można także zapisać w sposób bardziej czytelny:

$$\delta E^{(j)} = -\left[e_i^{(j)} - w_0^{(i)}\right]\, \delta y_i^{(j)}$$

Na podstawie tego wzoru rozważymy możliwe zachowania sieci. Weźmiemy pod uwagę jeden z neuronów sieci (o numerze i) rozważając wszystkie możliwe stany jego pobudzenia $e_i^{(j)}$ i sygnalu wyjściowego $y_i^{(j)}$.

Załóżmy, że w pewnym kroku j łączne pobudzenie neuronu $e_i^{(j)}$ przekracza próg $w_0^{(i)}$. Wówczas zgodnie z zasadą działania rozważanego modelu neuronu — na wyjściu tego neuronu powinien pojawić się sygnal $y_i^{(j)} = 1$. Oznacza to, że czynnik $\delta y_i^{(j)}$ musi być w takim przypadku dodatni lub zerowy — nigdy ujemny. Równocześnie przy $e_i^{(j)} > w_0^{(i)}$ także czynik w kwadratowym nawiasie musi być dodatni, a zgodnie z podanym wzorem zmiana calkowitej „energii" sieci $\delta E^{(j)}$ musi być ujemna lub zerowa.

Do podobnego wniosku dochodzimy przy przeciwnym założeniu. Jeśli $e_i^{(j)} < w_0^{(i)}$, to oczywiście $y_i^{(j)} = 0$ i oczywiście wtedy czynnik $\delta y_i^{(j)}$ musi być w takim przypadku ujemny lub zerowy — nigdy dodatni. W rezultacie także i w tym przypadku zmiana calkowitej „energii" sieci $\delta E^{(j)}$ musi być ujemna (lub zerowa).

Wreszcie gdy $e_i^{(j)} = w_0^{(i)}$, to oczywiście $\delta E^{(j)} = 0$ i energia sieci nie zmienia się.

Z tego prostego rozumowania wynika, że calkowita „energia" sieci może pozostawać stala lub może się zmniejszać — natomiast nie może rosnąć. Skoro w trakcie pracy sieci „energia" stale maleje — musi wreszcie osiągnąć stan odpowiadający minimum — lokalnemu albo globalnemu. Po osiągnięciu tego minimum dalsze zmiany w stanie sieci są niemożliwe i cały proces zatrzymuje się. Jak widać w takim przypadku sieć jest stabilna.

7.4 Procesy dynamiczne w sieciach Hopfielda

Dynamiczne właściwości sieci Hopfielda wygodnie jest rozważać na podstawie ciągłego modelu zachowania sieci. Wektor **sumarycznych pobudzeń** wszystkich neuronów sieci **e** można wtedy związać z wektorami sygnalów wyjściowych z elementów sieci **Y** oraz sygnalów wejściowych (zewnętrznych) **X** za pomocą macierzowego równania różniczkowego

$$\frac{d\mathbf{e}}{dt} = -\frac{\mathbf{e}}{\tau} + \mathbf{W}\,\mathbf{Y} + \mathbf{X}$$

uzupełnionego nieliniowym równaniem charakterystyki statycznej jednego elementu

$$y_i = \varphi(e_i)$$

Dla takiego nieliniowego systemu dynamicznego możliwe jest określenie funkcji Lapunowa w postaci

$$L = -\frac{1}{2}\,\mathbf{Y}^T\,\mathbf{W}\,\mathbf{Y} - \mathbf{X}^T\,\mathbf{Y} + \frac{1}{\tau}\sum_{i=1}^{k}\int_0^{y_i} \varphi^{-1}(\xi)\, d\xi$$

która upraszcza się w przypadku funkcji φ zbliżonej kształtem do skoku jednostkowego, przyjmując postać

$$L = -\frac{1}{2}\mathbf{Y}^T\mathbf{W}\mathbf{Y} - \mathbf{X}^T\mathbf{Y}$$

Łatwo zauważyć bliski i bezpośredni związek pomiędzy podaną funkcją, a podaną wyżej funkcją „energii" minimalizowanej przez sieć. Jest to kolejny argument przemawiający za tym, że procesy dynamiczne w sieci będą zawsze przebiegały w kierunku minimalizacji „energii", chociaż przebieg ten może być nie monotoniczny, a nawet mogą się pojawiać trudności z zagwarantowaniem zbieżności.

7.5 Pamięć autoasocjacyjna

Sieć Hopfielda może być wykorzystana jako tzw. **pamięć autoasocjacyjna** (skojarzeniowa). Czasem ten rodzaj sieci nazywany też bywa CAM od *Content Adressable Memory*. Rozważmy teraz w skrócie jej działanie. Załóżmy, że sieć powinna zapamiętać szereg wektorów $\mathbf{D}_j$ $(j = 1, 2, \ldots, M)$ i po pojawieniu się wektora wejściowego $\mathbf{X}$ *podobnego* do któregoś z zapamiętanych wzorców sieć powinna, na zasadzie swobodnych skojarzeń, odnaleźć i odtworzyć ten zapamiętany wektor $\mathbf{D}_j$, który kojarzy się z wektorem $\mathbf{X}$. Zakładamy, że wymiar wektora $\mathbf{X}$ oraz wymiary wszystkich wektorów $\mathbf{D}_j$ wynoszą k, czyli odpowiadają liczbie elementów (neuronów) wchodzących w skład sieci.

W praktyce odbywa się to w taki sposób, że uczy się sieć metodą **Hebba**, wytwarzając współczynniki wagowe $w_i^{(m)}$ przy połączeniach między i-tym i m-tym neuronem zgodnie z zasadą

$$w_i^{(m)} = \sum_{j=1}^{M} y_i^{(j)}\, y_m^{(j)}$$

gdzie $y_i^{(j)}$ jest i-tą składową wektora $\mathbf{D}_j$. W wyniku takiego postępowania słowy macierz $\mathbf{W}$ połączeń pomiędzy elementami sieci ma postać

$$\boldsymbol{W} = \sum_{j=1}^{M} \boldsymbol{D}_j^T\, \boldsymbol{D}_j$$

Działanie sieci polega na impulsowym (jednorazowym) podaniu sygnałów wejściowych $\mathbf{X}$ i swobodnej relaksacji sieci do najbliższego stanu stabilnego, odpowiadającego minimum funkcji „energii". Stan ten interpretować można jako „skojarzony" z bodźcem $\mathbf{X}$ zapamiętany sygnał $\mathbf{D}$.

Pojemność takiej pamięci szacowana jest przez różnych autorów rozmaicie. Jak wiadomo, sieć binarnych elementów złożona z N neuronów może znajdować się ogólnie w jednym z 2^N rozróżnialnych stanów, jednak rzeczywista pojemność pamięci jest znacznie mniejsza. **Hopfield** szacował liczbę stanów możliwych do zapamiętania w sieci na około $0,15\ N$. Podobne oszacowanie dali **Abu-Mostafa** i **St. Jaques**.

Przy opisanych wyżej zastosowaniach sieci neuronowych jako pamięci asocjacyjnych sygnały wyjściowe z elementów sieci przyjmuje się jako **ciągłe** w przedziale domkniętym od -1 do 1 $(y_m^{(j)} \in [-1, 1])$, a nieliniowa funkcja

$$y_m^{(j)} = \varphi\,(e_m^{(j)})$$

może być przedstawiona w formie klasycznej sigmoidy

$$y_m^{(j)} = \varphi\left(e_m^{(j)}\right) = \frac{1}{1+\exp\left(-\beta\, e_m^{(j)}\right)}$$

Funkcja ta dla dużych β jest bardzo stroma i przypomina funkcję progową, dyskutowaną wyżej. Natomiast dla małych β funkcja ta ma przebieg gładszy i łagodniejszy, w wyniku czego zachowanie sieci ciągłej zaczyna istotnie odbiegać od opisanego wyżej zachowania sieci dyskretnej. W szczególności przy malejącym β zaczynają znikać niektóre punkty stałe.

7.6 Maszyny Boltzmanna

Z opisywaną wyżej siecią Hopfielda kojarzone są zwykle tak zwane **Maszyny Boltzmanna**. Koncepcja takiej maszyny oparta jest na założeniu, że stan (sygnał wyjściowy $y_m^{(j)}$) każdego neuronu może się zmieniać w sposób **losowy** z określonym prawdopodobieństwem. Prawdopodobieństwo to zależy od „energii" i „temperatury" sieci podobnie jak w systemach fizycznych (termodynamicznych), w których gęstość prawdopodobieństwa $p\,(E,T)$ energii systemu E związana jest z temperaturą T znanym wzorem Boltzmanna

$$p\,(E,T) = e^{-E/kT}$$

gdzie k jest stałą Boltzmanna. Przenosząc to fizyczne prawo do informacyjnego systemu, jakim jest sieć neuronowa, możemy na każdym kroku j związać z neuronem o numerze m „energię" $E_m^{(j)}$ wyrażającą nadwyżkę jego łącznego pobudzenia $e_m^{(j)}$ ponad progiem pobudzenia $w_0^{(m)}$.

$$E_m^{(j)} = e_m^{(j)} - w_0^{(m)}$$

Następnie w oparciu o energię $E_m^{(j)}$ wyznaczane jest prawdopodobieństwo $p_m^{(j)}$ zgodnie z regułą będącą uogólnieniem **prawa Boltzmanna**:

$$p_m^{(j)} = 1 \,/\, \left[1+\exp\left(-\delta\, E_m^{(j)}\,/T^{(j)}\right)\right]$$

gdzie δ jest pewną arbitralnie dobieraną stałą, a $T^{(j)}$ reprezentuje symulowaną w j-tym kroku „temperaturę" sieci. Przy podanych założeniach algorytm doprowadzania sieci do stanu równowagi sprowadza się do kolejnego wykonywania dwóch kroków:

1. Dla ustalonego $T^{(j)}$ wyliczane są wszystkie wartości $p_m^{(j)}$, a następnie **losowo** z prawdopodobieństwem $p_m^{(j)}$ ustawiane są wartości sygnałów wyjściowych neuronów $y_m^{(j)}$. Konkretnie wykonuje się to w taki sposób, że dla każdego kroku j i dla każdego neuronu m losowana jest z równomiernym rozkładem prawdopodobieństwa wartość przypadkowa $\xi \in [0,1]$ (jest to zwykle liczba pseudoprzypadkowa pochodząca z generatora liczb pseudolosowych w przypadku realizacji komputerowej modelu sieci), a następnie ustala się wartości $y_m^{(j)}$ zgodnie z regułą:

$$y_m^{(j)} = \begin{cases} 1 & \text{gdy } \xi \le p_m^{(j)} \\ 0 & \text{w przeciwnym przypadku} \end{cases}$$

2. Obniża się stopniowo wartość $T^{(j)}$ w kolejnych krokach (na przykład według reguły

$$T^{(j+1)} = T^{(j)} - \varepsilon$$

albo

$$T^{(j+1)} = T^{(j)}(1 - \varepsilon)$$

i powtarza się punkt **1** aż do osiągnięcia stanu równowagi.

Proces ten — na podstawie analogii z procesem tzw. **odprężania** stosowanego w cieplnej obróbce metali nazywa się zwykle **symulowanym odprężaniem** *(simulated annealing)*, ponieważ podobnie jak obrabianemu materialowi — sieci nadaje się na początku wysoką „temperaturę" $T^{(j)}$, a potem stopniowo się ją obniża doprowadzając do osiągnięcia globalnego minimum łącznej energii wewnętrznej sieci.

Technika „maszyny Boltzmanna" może być zastosowana do dowolnej sieci, nie tylko sieci Hopfielda (autoasocjacyjnych). Jeśli sieć ma wyróżnione sygnały wejściowe i wyróżnione sygnały wyjściowe (a więc jest siecią typu *feedforward* zgodnie z wyżej przyjętą terminologią), to wówczas także można skorzystać z koncepcji osiągania przez sieć stanu „równowagi termodynamicznej" w taki sposób, że w kroku $j = 0$ podaje się na wejście sieci sygnały $\mathbf{X}$, a potem stosując wyżej opisany algorytm dla $j = 1, 2, \ldots$ doprowadza się sieć do stanu równowagi o najmniejszym potencjale energetycznym i odczytuje się odpowiedź z wyjść neuronów mających charakter elementów końcowej warstwy.

Opisany algorytm został uogólniony przez **Hintona** i **Sejnowskiego** [Hint86] w sposób następujący. Załóżmy, że mamy uczyć pewną konkretną sieć jakiejś określonej formy przetwarzania informacji, posługując się ciągiem uczącym $U = \{< \mathbf{X}^k, \mathbf{Z}^k >,\ k = 1, 2, \mathbf{N}\}$, gdzie $\mathbf{X}$ jest wektorem sygnałów wejściowych podawanych do **niektórych** elementów sieci, a $\mathbf{Z}$ jest wektorem wyjściowym określającym sygnały, jakie są oczekiwane na wyjściach także niektórych elementów sieci. Wówczas algorytm uczenia sieci można opisać wymieniając trzy czynności:

1. Obliczanie „związanych" prawdopodobieństw.

   Krok ten przewiduje wykonanie następujących operacji:

   a. Wymusza się wynikające z ciągu uczącego wartości wejść $\mathbf{X}^k$ i wyjść $\mathbf{Z}^k$ wyróżnionych neuronów wejściowych i wyjściowych.

   b. Pozwala się sieci dojść do stanu równowagi.

   c. Rejestruje się sygnały wyjściowe $y_m^{(k)}$ wszystkich elementów sieci.

   d. Powtarza się punkty od **a** do **c** dla wszystkich elementów ciągu uczącego (dla $k = 1, 2, \ldots, \mathbf{N}$), zbierając statystykę, dzięki której po zakończeniu pokazów wszystkich elementów ciągu uczącego możliwe jest obliczenie prawdopodobieństw P_{ij}^{+} tego, że sygnały wyjściowe neuronów o numerach i oraz j mają **równocześnie** wartość 1.

2. Obliczanie „nie związanych" prawdopodobieństw.

   Krok ten przewiduje wykonanie następujących operacji:

   a. Wymusza się **przypadkowe** wartości sygnałów wyjściowych wszystkich neuronów sieci.

b. Pozwala się sieci dojść do stanu równowagi.

c. Rejestruje się sygnały wyjściowe $y_m^{(k)}$ wszystkich elementów sieci.

d. Powtarza się punkty od **a** do **c** wielokrotnie, zbierając statystykę, dzięki której po zakończeniu pokazów wszystkich elementów ciągu uczącego możliwe jest obliczenie prawdopodobieństw P_{ij}^- tego, że sygnały wyjściowe neuronów o numerach i oraz j mają **równocześnie** wartość 1.

3. Na podstawie wartości P_{ij}^+ oraz P_{ij}^- koryguje się wartości współczynników wagowych $w_i^{(j)}$ oraz $w_j^{(i)}$ łączących neurony o numerach i oraz j. Wartości te są zwiększane o wartość δ_{ij} wyliczaną ze wzoru

$$\delta_{ij} = \eta \left[P_{ij}^+ - P_{ij}^-\right]$$

Stosując opisany wyżej algorytm można — zwykle po kilku iteracjach — znaleźć optymalne wartości współczynników wagowych zapewniających realizację zamierzonej (zadanej ciągiem uczącym) zależności pomiędzy sygnałami wejściwymi i wyjściowymi w dowolnie dużej sieci.

7.7 Przykład zastosowania sieci Hopfielda — konwerter

Po przedstawionych wyżej rozważaniach ogólnych warto przedstawić kilka przykładów zastosowań opisywanych sieci. W pracy Tanka i Hopfielda [Tank86] opisano realizację za pomocą sieci typu Hopfielda przetwornika analogowo-cyfrowego. Rolę neuronów w opisywanym systemie odgrywały wzmacniacze operacyjne połączone w ten sposób, że wyjście każdego z nich podawane było na wejścia wszystkich pozostałych przez regulowane rezystory pełniące rolę współczynników wagowych[1]. Zapis $w_i^{(j)}$ jest tu rozumiany jako waga i-tego wejścia w neuronie o numerze j. Ponieważ wzmacniacze posiadały wejścia odwracające fazę i wejścia proste — możliwe było realizowanie zarówno wartości $w_i^{(j)} > 0$ jak i $w_i^{(j)} < 0$. Dla zapewnienia stabilności działania sieci wykluczono połączenia łączące wyjście danego neuronu z jego własnym wejściem ($w_i^{(i)} = 0$) oraz zapewniono symetrię sieci ($w_i^{(j)} = w_j^{(i)}$). Dodatkowo na wejścia wszystkich elementów sieci podawany jest **ten sam** sygnał wejściowy $\mathbf{X}$ przez rezystory odpowiadające wadze $w_x^{(j)}$. Zadaniem sieci jest wytworzenie na wyjściach $y^{(j)}$ wszystkich neuronów ($j = 1, 2, \ldots, k$) sygnałów $y^{(j)}$ odpowiadających binarnemu k-bitowemu kodowi sygnalizującemu analogową wartość $\mathbf{X}$. Sygnały $y^{(j)}$ powinny być przy tym binarne (czyli powinny przyjmować wyłącznie wartości 0 lub 1), a wartość liczby dwójkowej reprezentowanej przez wartości $y^{(j)}$ (wynosząca $\sum_{j=0}^{k-1} 2^j\, y^{(j)}$) powinna być jak najbliższa wartości $\mathbf{X}$.

W takiej sieci poprawna praca systemu jest zapewniona, jeśli dokonana jest minimalizacja funkcji „energii" o postaci

$$E = -1/2 \left[X - \sum_{j=0}^{k-1} 2^j\, y^{(j)}\right]^2 + \sum_{j=0}^{k-1} \left(2^{2j-1}\right) \left[y^{(j)} \left(1 - y^{(j)}\right)\right]$$

[1] Dokładniej — elektrycznym analogonem wartości wagi jest w rozważanym systemie przewodność określonego rezystora wejściowego.

Jak wiadomo działanie sieci polega na minimalizowaniu funkcji „energii". Efekt tego działania w rozważanej sieci łatwo sobie wyobrazić. Pierwszy składnik przytoczonego wzoru może być traktowany jako suma kwadratów błędów popełnianych przez sieć (rozbieżność między wartością **X**, a wartością binarnego kodu formowanego przez sygnały $y^{(j)}$). Drugi składnik natomiast osiąga małe wartości przy $y^{(j)}$ przyjmujących wartości binarne (0 lub 1), natomiast przy innych $y^{(j)}$ wartość tego składnika wzrasta stanowiąc „karę" za niewłaściwe wartości wyjść sieci. W sumie minimalizacja wskazanej funkcji energii doprowadzi do tego, że na wyjściach sieci pojawią się sygnały $y^{(j)}$ zbliżone do wartości binarnych kodujące (zgodnie z zasadami arytmetyki dwójkowej) wartość wejściowego sygnału **X**. Jak wynika z porównania przytoczonego wyżej wzoru określającego funkcję „energii" w rozważanej sieci z wcześniej opisanym wzorem ogólnym — współczynniki wagowe ustalone w rozważanej sieci wyrazić można wzorami:

$$w_i^{(j)} = -2^{(i+j)}$$
$$w_x^{(j)} = 2^j$$

Udana budowa i efektywna eksploatacja opisanej sieci [Tank86] potwierdziły, że sieci Hopfielda mogą być użyte do realizacji konkretnych zadań przetwarzania informacji — chociaż oczywiście przetwornik A/C można zbudować znacznie prościej bez korzystania z sieci.

7.8 Rozwiązywanie problemu komiwojażera

Podobny charakter — efektownego zastosowania sieci do rozwiązania zadania możliwego do rozwiązania także innymi metodami, przyniosła słynna praca **Van den Bouta** i **Millera** [Bout88], pokazująca zastosowanie sieci Hopfielda do rozwiązania klasycznego problemu optymalizacyjnego — tak zwanego „problemu komiwojażera", oznaczanego zwykle jako TSP *(Traveling Salesman Problem)*. Problem ten polega na ustaleniu optymalnej trasy objazdu n miast przez wędrownego sprzedawcę, który musi być we wszystkich miastach przynajmniej raz i chce oczywiście wydać jak najmniej na same podróże. Jako dane przy rowiązywaniu problemu podane są odległości d_{ij} pomiędzy wszystkimi miastami, a koszt podróży sprzedawcy równy jest długości sumarycznej przebytej przez niego drogi.

Problem ten — jak udowodnili **Garley** i **Johnson** w 1979 roku — należy do zadań NP-zupełnych, czyli czas jego rozwiązywania rośnie wykładniczo przy wzroście liczby rozważanych miast n. Istotnie, przy n miastach możliwe jest zbudowanie $D = n!/(2n)$ rozróżnialnych tras. Nawet przy niewielkich wartościach n jest bardzo dużo ewentualności do rozważenia i zbadania (np. dla $n = 60$ $D = 69,34155\ 10^{78}$) licznych zadań optymalizacji kombinatorycznej i dlatego metody rozwiązywania problemu komiwojażera są w centrum uwagi badaczy zajmujących się metodami optymalizacji i teorią badań operacyjnych. Istnieje uzasadnione przypuszczenie, że metody zastosowane do rozwiązywania problemu komiwojażera mogą niemal natychmiast znaleźć zastosowanie przy rozwiązywaniu innych problemów kombinatorycznych. Opisana niżej technika rozwiązywania tego problemu z wykorzystaniem sieci neuronowej typu Hopfielda nie jest wolna od wad — w szczególności przy jego rozwiązywaniu sieć łatwo ulega „wciąganiu" przez lokalne minima, co prowadzi do rozwiązań suboptymalnych. Jednak zdarza się to stosunkowo rzadko (w pracach Tanka [Tank85] referowane są eksperymenty, podczas których sieć rozwiązująca problem TSP dla 10 miast w 16 przypadkach na 20 podejmowanych prób znalazła optymalną marszrutę (wybraną spośród 181 440

możliwych!), a ponadto nowsze prace Van den Bouta i Millera [Bout88] proponują wybór efektywniejszej sieci i lepszego alagorytmu poszukiwania punktu równowagi.

Równocześnie zalety „neuronowego" rozwiązania są bezsporne — pracując współbieżnie neurony sieci mogą rozwiązać postawiony problem w krótkim czasie mimo jego niewielomianowej złożoności. Co więcej — wzrost wymiaru problemu będzie przy takiej realizacji wymagał rozbudowy sieci, ale nie będzie powodował wydłużenia czasu obliczeń[2], co jest perspektywą bardzo zachęcającą z punktu widzenia licznych zastosowań.

Kluczem do sukcesu przy stosowaniu sieci neuronowej w problemie TSP jest odnalezienie odpowiedniej reprezentacji danych. W opisanym przez Tanka rozwiązaniu problemu każde miasto reprezentowane jest za pomocą wiersza zawierającego n neuronów. W takim wierszu **dokładnie jeden** neuron powinien przyjmować wartość „1", a **wszystkie pozostałe** mają sygnały wyjściowe odpowiadające wartości „0". Pozycja (od 1 do n), na której występuje neuron sygnalizujący „1" odpowiada kolejności, w jakiej to właśnie miasto ma być odwiedzone przez wędrownego sprzedawcę. Tak więc jedynka na pierwszej pozycji oznacza miasto, które komiwojażer powinien odwiedzić jako pierwsze, jedynka na drugiej pozycji sygnalizuje drugie w kolejności odwiedzane miasto itd. Oczywiście z opisu tego wynika, że liczba wierszy musi odpowiadać liczbie rozważanych miast (czyli musi wynosić n), zatem łączna liczba potrzebnych neuronów wynosi n^2.

Każdy neuron w tej sieci musi być indeksowany **dwoma** wskaźnikami. Pierwszy z nich dotyczy numeru miasta, a drugi kolejności, w jakiej to miasto powinno być odwiedzane. Tak więc w tej sieci sygnał y_{xi} oznaczać będzie sygnał wyjściowy z neuronu wchodzącego w skład wiersza odpowiadającego miastu numer $\times$, przy czym neuron ten odpowiada i-tej pozycji w tym wierszu, czyli $y_{xi} = 1$ oznacza, że x-te miasto należy odwiedzić w i-tej kolejności. Opisując funkcję „energii" minimalizowanej przez rozważaną sieć trzeba brać pod uwagę cztery jej składniki:

$$E_1 = A/2 \sum_x \sum_i \sum_{i \neq j} y_{xi}\, y_{xj}$$

$$E_2 = B/2 \sum_i \sum_x \sum_{z \neq x} y_{xi}\, y_{zj}$$

$$E_3 = C/2 \left[\left(\sum_x \sum_i y_{xi} \right) - n \right]^2$$

$$E_4 = D/2 \sum_x \sum_{z \neq x} \sum_i d_{xz}\, y_{xi}\, (y_{z,i+1} + y_{z,i-1})$$

Składniki te mają następującą interpretację. E_1 ma zerową wartość wtedy i tylko wtedy, jeśli w każdym wierszu jest najwyżej jedna jedynka. Składnik ten można więc traktować jako „karę" za niedotrzymanie jednego z podstawowych warunków zadania — że każde miasto ma mieć jednoznacznie określoną kolejność odwiedzin przez komiwojażera. Składnik E_2 ma zerową wartość wtedy i tylko wtedy, gdy w każdej kolumnie (oznaczającej konkretny etap podróży) będzie najwyżej jedna jedynka. Jest to więc „kara" za naruszenie warunku jednoznacznego określenia, kiedy jakie miasto należy odwiedzić. Składnik E_3 jest zerem wtedy i tylko wtedy, gdy w macierzy będzie dokładnie n jedynek. Wreszcie składnik E_4

[2] Oczywiście w większej sieci na ogół dłużej będzie trwało dochodzenie do stanu równowagi, przeto w jakimś stopniu złożoność problemu będzie wpływała na wydłużenie czasu obliczeń — jednak oczywiście w stopniu nieporównanie mniejszym, niż ma to miejsce w typowych, szeregowych komputerach.

oznacza długość wybranej drogi; przy obliczaniu wartości tego składnika wszystkie wskaźniki przy y są brane modulo $n(y_{n+j} = y_j)$.

Współczynniki A, B, C i D są wybierane arbitralne i oznaczają względne „wagi" poszczególnych warunków. Konkretnie duże wartości A, B i C oznaczają silne związanie poszukiwanych rozwiązań z warunkami zadania, natomiast duże wartości D oznaczają silne związanie poszukiwanego rozwiązania z optymalizowaną funkcją celu (minimalizacją kosztu podróży). Dobór konkretnych wartości stałych A, B, C i D musi być wynikiem poszukiwań o charakterze „prób i błędów", gdyż w dostępnej literaturze brak konkretnych rad na temat ich doboru. Jednak, by nie pozostawiać Czytelnika bez żadnej informacji, można dodać, że w pakiecie programów modelujących sieci neuronowe o nazwie *NeuralWorks Professional II/PLUS* firmy *NeuralWare, Inc.* używane są następujące parametry:

$$A \cong B \cong D \cong 1000$$

Wartość C dobierana jest w zależności od relacji pomiędzy parametrem n użytym we wzorze dla składnika E_3, a rzeczywistą liczbą miast, którą dla odróżnienia oznaczmy chwilowo jako n^* (patrz niżej). Heurystyka przyjmowana przez firmę *NeuralWare* nakazuje stosowanie zależności

$$C = \frac{3}{4} D\,(n - n^*)$$

W oryginalnej publikacji Hopfielda i Tanka proponowano

$$A = B = D = 500$$

oraz

$$C = 200$$

jednak opisanych w tej pracy wyników nie udało się powtórzyć innym badaczom, zatem trudno te parametry uznać za szczególnie godne polecenia. Do zagadnienia tego powrócimy dalej.

W rozważanej sieci współczynniki wagowe określające parametry połączeń pomiędzy i-tym neuronem x-tej warstwy, a j-tym neuronem z-tej warstwy wyrażają się wzorem:

$$w_{xi,zj} = -A\,\delta_{xz}(1 - \delta_{ij}) - B\,\delta_{ij}\,(1 - \delta_{xz}) - C - D\,d_{xz}\,(\delta_{j,i+1} + \delta_{j,i-1})$$

gdzie δ_{ij} oznacza funkcję Kroneckera

$$\delta_{ij} = \begin{cases} 1 & \text{gdy i=j} \\ 0 & \text{w przeciwnym przypadku} \end{cases}$$

Dodatkowym elementem parametryzującym sieć jest zestaw wyrazów wolnych (progów pobudzenia), których wartości dla wszystkich elementów sieci są identyczne i wynoszą

$$w^0_{xz} = Cn$$

Dość istotne znaczenie ma problem poprawnego skalowania wszystkich odległości d_{xz}, występujących w cytowanych wzorach. Zalecane jest, aby odległości te przyjmowały wartości w ustalonym przedziale $[0, 1]$ ze średnią wartością poniżej 0,5. W cytowanych już wyżej

opracowaniach firmy *NeuralWare* podkreślono, że optymalne wyniki uzyskano przyjmując średnią wartość zadawanych odległości d_{xz} równą 0,375.

W cytowanych wyżej pracach Tanka i Hopfielda stosowano nieliniową funkcję opisującą neuron

$$y_{xz} = \varphi(e_{xz})$$

w postaci

$$y_{xz} = 1/2\,[1 + \tanh(e_{xz}/e_0)]$$

Charakterystyka ta zależy od parametru e_0, który reguluje jej kształt, przyjęcie zbyt dużej wartości e_0 prowadzi do ustalania się w sieci stanów końcowych, którym odpowiadają wartości y_{xz} nie będące wartościami bliskimi 0 albo 1, a więc rozwiązania są niejednoznaczne, natomiast za małe wartości e_0 utrudniają znalezienie optymalnych wartości wyznaczających najlepszą trasę komiwojażera. Warto odnotować, że dla $e_0 \to 0$ funkcja $\varphi(e_{xz})$ staje się bardzo stroma i praktycznie przechodzi w funkcję skoku jednostkowego. Hopfield w swoich pracach używał wartości $e_0 = 0,2$.

Dynamika sieci rozwiązującej zadanie TSP może być opisana układem równań różniczkowych, opisujących zachodzące w czasie zmiany sygnałów wyjściowych y_{xi} wszystkich neuronów sieci oraz ich sumarycznych pobudzeń e_{xi}.

$$\frac{de_{xi}}{dt} = -\frac{e_{xi}}{\tau} - A\sum_{i\neq j} y_{xj} - B\sum_{z\neq x} y_{zi} - C\left[\left(\sum_x \sum_j y_{xj}\right) - n\right]^2 - D\sum_{z\neq x} d_{xz}\,(y_{z,i+1} + y_{z,i-1})$$

rozważanych oczywiście wraz z równaniami algebraicznymi ustalającymi wartości sygnałów wyjściowych

$$y_{xi} = 1/2\,[1 + \tanh(e_{xi}/e_0)]$$

Na wyniki uzyskiwane przy rozwiązywaniu problemu komiwojażera duży wpływ mogą mieć warunki początkowe, przyjmowane dla neuronów sieci przy jej startowaniu. Ważne są także współczynniki występujące we wzorze opisującym funkcję „energii", które niekiedy trzeba ustawiać w sposób nieco zaskakujący. Na przykład w składniku E_3 występuje parametr n, który odpowieda liczbie miast, które ma odwiedzić komiwojażer. Otóż Hopfield i Tank w swojej publikacji infomują, że dla uzyskania zbieżnego procesu uczenia przyjmowali n większe od rzeczywistej liczby miast n^* (konkretnie $n = 1,5n^*$), przy czym nie jest dane żadne uzasadnienie dla takiego postępowania. Ten fakt, a także rozbieżności uzyskiwane przez innych badaczy usiłujących powtórzyć eksperymenty Hopfielda sprawiły, że klasyczna praca [Hopf85] poddawana była wielokrotnie modyfikacjom. Między innymi druzgocącej krytyce metodę Hopfielda i Tanka poddali **Aiyer, Niranjan** i **Fallside** [Aiye90]. Badania symulacyjne przeprowadzone przez autora wydają się potwierdzać krytyczne uwagi, sformułowane przez tych autorów. Konkretnie badania te wykazały, że mimo wielu prób nie udało się nigdy uzyskać tak dobrych wyników, jak opisywane przez Hopfielda i Tanka, chociaż odtwarzano pieczołowicie wszystkie podane w ich publikacji parametry sieci. Natomiast wprowadziwszy modyfikacje proponowane przez Aiyera i współpracowników uzyskano bardzo dobre wyniki (zarówno w sensie dokładności i optymalności podawanych przez sieć rozwiązań jak i w sensie znacznie mniejszych czasów działania sieci), co dowodzi, że nawet do publikacji największych „klasyków" należy podchodzić w sposób krytyczny i rozważny. Do zagadnienia tego powrócimy w rozdziale 10.2.

Rozdział 8

Sieci pamięci skojarzeniowej

8.1 Sieć Hintona

Pamięć skojarzeniowa jest jednym z podstawowych atrybutów ludzkiego mózgu. Pamięć taka ma dwie istotne cechy, odróżniające ją w sposób zasadniczy od pamięci systemów technicznych, na przykład komputerów. Po pierwsze, jak wynika z jej nazwy, informacje zarejestrowane w pamięci asocjacyjnej mogą być dostępne poprzez podanie na wejściu systemu informacji skojarzonej, selekcjonującej jedną z zapamiętanych wiadomości **na drodze asocjacji**, a więc na drodze zdecydowanie odmiennej od dostępu adresowego, charakterystycznego dla wszelkich klasycznych systemów obliczeniowych, baz danych czy nawet systemów ekspertowych. Po drugie, ślad pamięciowy, zwany czasem *engramem*, nie ma w pamięci asocjacyjnej ścislej lokalizacji, gdyż każda zarejestrowana informacja zlokalizowana jest w istocie w **całej** pamięci, na zasadzie kolektywnego działania wszystkich jej elementów. Taka technika zapisu informacji nazywana jest *hololograficzną*. Jej fizycznym przykładem jest **hologram**, do którego często odwołują się badacze zafascynowani właściwościami ludzkiej pamięci i ludzkiego sposobu uczenia się.

O zaletach i wadach pamięci skojarzeniowej napisano już całe tomy, nie zachodzi więc potrzeba dyskusji tych podstawowych zagadnień w tej książce, natomiast niewątpliwie interesująca jest tu możliwość realizacji pamięci skojarzeniowej za pomocą sieci neuronowych. Najprostszą siecią tego typu jest sieć opracowana w 1981 roku przez **Hintona** [Hint81] i tę sieć najpierw niżej zaprezentujemy.

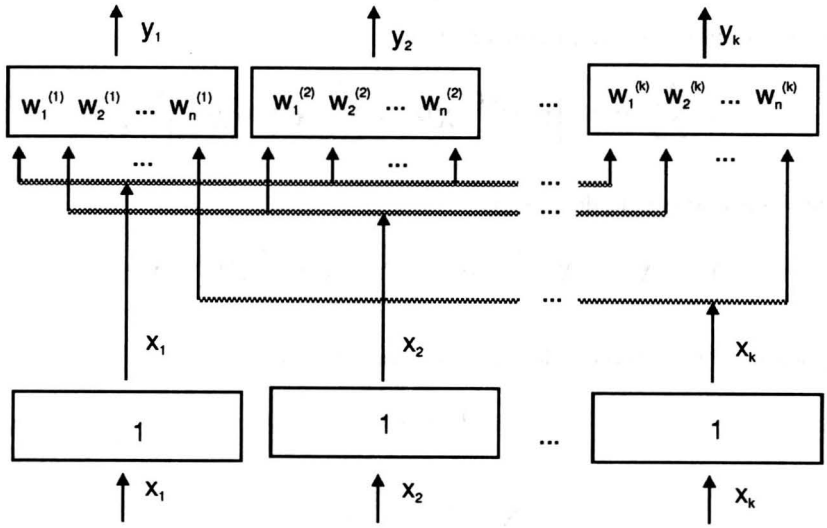
Rozważymy sieć o dwóch warstwach z pełnym („każdy z każdym") schematem połączeń elementów pierwszej warstwy z elementami drugiej warstwy. Niech sygnały wejściowe podawane do pierwszej warstwy tworzą wektor $\mathbf{X}$, a sygnały wyjściowe, powstające na wyjściach drugiej warstwy, tworzą wektor $\mathbf{Y}$.

Jak zauważymy, pierwsza warstwa nie uczestniczy właściwie w procesie przetwarzania informacji i pełni raczej funkcję elementu normalizującego sygnały.

Cały sens działania sieci polega więc na odpowiednim doborze wag $w_i^{(j)}$, gdyż w nich zawierać się musi cała wiedza systemu. Wiedza ta pochodzi z ciągu uczącego $\mathbf{U}$, którego struktura może być rozważana jako ciąg podlegających kojarzeniu par wektorów: wejściowego $\mathbf{X}$ i wyjściowego $\mathbf{Y}$.

Niech ciąg ten zostanie zapisany w następujący sposób:

$$U \leftarrow \{< X^{(1)}, Y^{(1)} >, < X^{(2)}, Y^{(2)} >, \ldots$$
$$\ldots, < X^{(k)}, Y^{(k)} >, < X^{(k+1)}, Y^{(k+1)} >, \ldots, < X^{(N)} Y^{(N)} >\}$$



Zakladając, że sieć podlega uczeniu metodą Hebba możemy stwierdzić, waga $w_i^{(j)}$ w kolejnych krokach procesu uczenia zmienia się zgodnie z regułą

$$w_i^{(j)(k+1)} = w_i^{(j)(k)} + \eta\, x_i^{(k)}\, y_i^{(k)}$$

Dla całej macierzy wag $\mathbf{W}$ obowiązuje zatem rekurencyjna reguła

$$W^{(k+1)} = W^{(k)} + \eta\, X^{(k)} \left[Y^{(k)}\right]^T$$

a po wykonaniu wszystkich N kroków procesu uczenia mamy macierz wag $\mathbf{W}$ określoną wzorem

$$\mathbf{W} = \sum_{k=1}^{N} \mathbf{X}^{(k)} \left[\mathbf{Y}^{(k)}\right]^T$$

Powyższy wzór jest prawdziwy pod warunkiem, że wszystkie wagi w sieci przed rozpoczęciem uczenia były zerowe ($\mathbf{W}^{(1)} = 0$) oraz przy założeniu, że $\eta = 1$.

Asocjacyjne i pamięciowe właściwości omówionej sieci mogą być prosto wykazane przy założeniu, że wektory wejściowe (wywołujące skojarzenia) będą ortonormalne, to znaczy

$$\mathbf{X}^{(i)} \left[\mathbf{X}^{(j)}\right]^T = \begin{cases} 1 & \text{gdy} \quad i = j \\ 0 & \text{gdy} \quad i \neq j \end{cases}$$

W takim wypadku po pojawieniu się na wejściu nauczonej sieci sygnałów odpowiadających dokładnie pewnemu wektorowi $\mathbf{X}$, który pojawił się w trakcie procesu uczenia (oznaczmy ten wektor $\mathbf{X}^{(j)}$), wówczas sygnał wyjściowy uzyskany z sieci można precyzyjnie obliczyć jako równy

$$\boldsymbol{Y} = \boldsymbol{W}\ \boldsymbol{X}^{(j)} = \sum_{k=1}^{N} \boldsymbol{X}^{(k)}\ \left[\boldsymbol{Y}^{(k)}\right]^T\ \boldsymbol{X}^{(j)}$$

Ale zgodnie z zasadami rachunku macierzowego

$$\boldsymbol{Y} = \sum_{k=1}^{N} \boldsymbol{X}^{(k)}\ \left[\boldsymbol{Y}^{(k)}\right]^T\ \boldsymbol{X}^{(j)} = \sum_{k=1}^{N} \boldsymbol{X}^{(k)}\ \left[\boldsymbol{X}^{(j)}\right]^T\ \boldsymbol{Y}^{(k)}$$

co można łatwo przekształcić do postaci

$$\boldsymbol{Y} = \boldsymbol{X}^{(j)}\ \left[\boldsymbol{X}^{(j)}\right]^T\ \boldsymbol{Y}^{(j)} + \sum_{k \neq j} \boldsymbol{X}^{(k)}\ \left[\boldsymbol{X}^{(j)}\right]^T\ \boldsymbol{Y}^{(k)}$$

zaś to na podstawie przyjętych założeń można zapisać jako

$$\boldsymbol{Y} = \boldsymbol{Y}^{(j)}$$

ponieważ

$$\mathbf{X}^{(j)}\ \left[\mathbf{X}^{(j)}\right]^T = 1$$

oraz

$$\sum_{k \neq j} \mathbf{X}^{(k)}\ \left[\mathbf{X}^{(j)}\right]^T\ \mathbf{Y}^{(k)} = 0$$

Jak z tego wynika, sieć Hintona po nauczeniu metodą Hebba pobudzona sygalem $\mathbf{X}^{(j)}$ odtwarza na swoim wyjściu dokładnie skojarzony z tym sygnalem (podczas uczenia) sygnal wyjściowy $\mathbf{Y}^{(j)}$.

8.2 Ogólne własności sieci Hintona

Opisany na końcu poprzedniego podrozdzialu wynik polegający na idealnym odtwarzaniu przez sieć zapamiętanych informacji — jest interesujący, ale ma raczej akademickie znaczenie, ponieważ warunek ortonormalności wejść jest w praktyce bardzo trudny do zachowania. Na szczęście dalsze analizy zachowania sieci Hintona wykazały, że warunek ortonormalności wejść można znacznie osłabić bez utraty możliwości sensownego korzystania z pracy sieci.

Podstawowe znaczenie dla tych rozważań miały prace Kohonena [Koho81], który rozpatrywał własności sieci autoasocjacyjnych i uzyskał bardzo ogólne wyniki, pozwalające określić warunki zapamiętywania i poprawnego odtwarzania informacji rejestrowanych przez sieć asocjacyjną. Kohonen stosował swój własny system oznaczeń, ale w tej książce jego osiągnięcia zostaną przedstawione z wykorzystaniem symboli ujednoliconych z innymi autorami. Rozważać będziemy sieć jednowarstwową o n wejściach i k wyjściach (liczba wyjść jest oczywiście identyczna z liczbą neuronów w sieci). Do każdego neuronu doprowadzone są wszystkie sygnały wejściowe, tworzące wektor $\mathbf{X} =< x_1, x_2, \ldots, x_n >$ oraz sygnał f_i

(gdzie i jest numerem neuronu), będący „wymuszeniem" powodującym określone zachowanie sieci podczas procesu uczenia. Sygnał wyjściowy z neuronu o numerze i, wyznaczyć można z równania

$$y_i = \sum_{j=1}^{n} w_{ij}\, x_j + f_i$$

zaś proces uczenia prowadzony według zmodyfikowanego algorytmu Hebba opisuje równanie różniczkowe

$$\frac{d}{dt} w_{ij} = \eta\, f_i\, x_j$$

Oczywiście podane wyżej zależności dotyczą wszystkich neuronów ($i = 1, 2, \ldots, k$) i mogą być zapisane w formie wektorowej

$$\boldsymbol{Y} = \boldsymbol{W}\,\boldsymbol{X} + \boldsymbol{F}$$

$$\frac{d}{dt}\boldsymbol{W} = \eta\, \boldsymbol{F}\, \boldsymbol{X}^T$$

Rozwiązanie powyższego równania wektorowego, przy założeniu że $\mathbf{X}$ i $\mathbf{F}$ pozostają niezmienne, ma postać

$$\boldsymbol{W}(t) = \eta\, t\, \mathbf{F}\, \mathbf{X}^T$$

Wyobraźmy sobie teraz proces uczenia polegający na prezentacji ciągu uczącego złożonego z par wektorów $\mathbf{X}^{(k)}$ i $\mathbf{F}^{(k)}$:

$$\boldsymbol{U} = \{< \boldsymbol{X}^{(1)}, \boldsymbol{F}^{(1)} >, < \boldsymbol{X}^{(2)}, \boldsymbol{F}^{(2)} >, \ldots, < \boldsymbol{X}^{(N)}, \boldsymbol{F}^{(N)} >\}$$

przy czym zakładamy, że każda para $< \mathbf{X}^{(k)}, \mathbf{F}^{(k)} >$ prezentowana jest sieci w ciągu odcinka czasu $\tau = 1/\eta$. Wówczas po pełnym cyklu uczenia macierz wag $\mathbf{W}$ wyraża się wzorem

$$\boldsymbol{W} = \sum_{k=1}^{N} \boldsymbol{X}^{(k)} \left(\boldsymbol{F}^{(k)}\right)^T$$

Po zakończeniu cyklu uczenia sygnały wymuszające nie występują ($F = 0$), natomiast sygnały wyjściowe z sieci są warunkowane wyłącznie przez jej sygnały wejściowe:

$$\mathbf{Y} = \mathbf{W}\,\mathbf{X}$$

Oczekujemy, że sygnały te będą zbliżone do narzucanych w trakcie procesu uczenia, ale **pewność**, że tak będzie, mamy tylko dla ortonormalnych sygnałów $\mathbf{X}^{(k)}$, co było wyżej dyskutowane. Rozważmy jednak sytuacje o łagodniejszych wymaganiach odnośnie struktury sygnałów $\mathbf{X}^{(k)}$ w trakcie procesu uczenia. Przy $\mathbf{X}^{(k)}$ *liniowo niezależnych* możemy zapisać wynikową macierz wag po zakończeniu procesu uczenia jako

$$\boldsymbol{W} = \boldsymbol{F}^* \left(\boldsymbol{X}^{*T}\, \boldsymbol{X}^*\right)^{-1} \boldsymbol{X}^{*T}$$

gdzie przez $\mathbf{F}^*$ i $\mathbf{X}^*$ oznaczono zagregowane do postaci macierzowej wektory występujące w ciągu uczącym. Ich budowa jest następująca:

$$\boldsymbol{F}^* = \left[\boldsymbol{F}^{(1)} \boldsymbol{F}^{(2)} \ldots \boldsymbol{F}^{(N)}\right]$$

$$\boldsymbol{X}^* = \left[\boldsymbol{X}^{(1)} \boldsymbol{X}^{(2)} \ldots \boldsymbol{X}^{(N)}\right]$$

Jeśli jednak wektory $\mathbf{X}^{(k)}$ nie są *liniowo niezależne*, to wówczas macierz wag $\mathbf{W}$ może być wyznaczona w postaci

$$\boldsymbol{W} = \boldsymbol{F}^* \boldsymbol{X}^{*+}$$

gdzie $\mathbf{X}^{*+}$ jest macierzą pseudoodwrotną w stosunku do $\mathbf{X}^*$. W przypadku liniowo niezależnych wektorów $\mathbf{X}^{(k)}$ macierz pseudoodwrotna $\mathbf{X}^{*+}$ jest identyczna z macierzą transponowaną $\mathbf{X}^{*T}$

$$\boldsymbol{X}^{*+} = \boldsymbol{X}^{*T}$$

i wówczas macierz $\mathbf{W}$ może być rozpatrywana jako macierz korelacji

$$\boldsymbol{W} = \boldsymbol{F}^* \boldsymbol{X}^{*T}$$

Tak więc sieć po nauczeniu rejestruje w swojej pamięci korelacyjne związki między sygnalem wejściowym $\mathbf{X}$ i wymuszającym $\mathbf{F}$. Brak liniowej niezależności wejść powoduje, że zamiast idelanej macierzy korelacji uzyskujemy w wyniku procesu uczenia jej optymalną (w sensie najmniejszych kwadratów) aproksymację.

Jeśli rozważy się ważny z praktycznego punktu widzenia przypadek szczególny $\mathbf{F}^* = \mathbf{S}^*$, wówczas

$$\boldsymbol{W} = \boldsymbol{F}^* \boldsymbol{F}^{*+}$$

i operacja wykonywana przez sieć nazywana jest *ortogonalną projekcją*, a jej interpretacja jest następująca: Dane w ciągu uczącym $\mathbf{N}$ wektorów $\mathbf{F}^{(k)}$ rozpina w k-wymiarowej przestrzeni pewną podprzestrzeń liniową L. Każdy konkretny wektor $\mathbf{F}$ może być wtedy poddany rzutowaniu na tę podprzestrzeń, w wyniku czego może być rozłożony na sumę dwóch wektorów

$$\mathbf{F} = \hat{\mathbf{F}} + \tilde{\mathbf{F}}$$

gdzie

$$\hat{\mathbf{F}} = \mathbf{F}^* \mathbf{F}^{*+} \mathbf{F}$$

jest efektem rzutowania wektora $\mathbf{F}$ na podprzestrzeń L, a wektor $\tilde{\mathbf{F}}$ może być traktowany jako *resisuum* — zależnie od interpretacji jest to wektor błędu (gdy celem jest optymalne dokonanie rzutowania) lub wektor nowości, jeśli celem funkcjonowania sieci ma być wykrycie rozbieżności pomiędzy sygnalem aktualnie pojawiającym się na wejściu sieci, a wektorami zapamiętanymi w trakcie procesu uczenia.

Optymalne rzutowanie ortogonalne ma miejsce w przypadku liniowo niezależnych wektorów wejściowych $\mathbf{X}^{(k)}$, gdy macierz $\mathbf{W}$ jest w istocie macierzą autokorelacji sygnałów $\mathbf{F}$:

$$\boldsymbol{W} = \boldsymbol{F}^* \boldsymbol{F}^{*T}$$

Ponieważ istnieje związek między iloczynem $\mathbf{F}^* \mathbf{F}^{*+}$ i $\mathbf{F}^* \mathbf{F}^{*T}$ dany jest wzorem

$$\boldsymbol{F}^* \boldsymbol{F}^{*+} = \alpha \sum_{j=0}^{\infty} \boldsymbol{F}^* \boldsymbol{F}^{*T} \left(\boldsymbol{I} - \alpha \boldsymbol{F}^* \boldsymbol{F}^{*T}\right)^k$$

zatem można uważać, że $\mathbf{F}^* \mathbf{F}^{*T}$ jest przybliżeniem zerowego rzędu (uwzględniającym tylko składnik dla $k = 0$) operatora $\mathbf{F}^* \mathbf{F}^{*+}$.

8.3 Dwukierunkowa pamięć asocjacyjna — sieć BAM

Rozwinięciem omówionych wyżej koncepcji Hopfielda i Hintona jest BAM — *Bidirectionxl Associxtive Memory* czyli dwukierunkowa pamięć skojarzeniowa. Koncepcja tej sieci wywodzi się z wczesnych prac Grossberga [Gros82], ale najważniejsze prace na temat BAM przedstawili **Kosko** i **Guest** [Kosk87]. Wykorzystaniem praktycznym sieci typu BAM zajmuje się przedsiębiorstwo o nazwie *Verac Corporation*. Sieć BAM jest częściowo oparta na koncepcji sieci ART, gdyż kluczem do jej działania jest wzajemne oddziaływanie sygnałów wejściowych i wyjściowych w zamkniętej pętli sprzężenia zwrotnego. Jednak cechy charakterystyczne sieci BAM są na tyle odmienne od sieci ART, że zwykle rozważa się te dwie sieci całkowicie oddzielnie.

W sieci BAM występują dwa wektory sygnałów, oznaczane zwykle jako $\mathbf{X}$ i $\mathbf{Y}$. Nie muszą one być jednakowych rozmiarów, przeciwnie zwykle zakłada się, że wektor $\mathbf{X}$ zawiera m składowych ($\mathbf{X} \in \mathcal{R}^m$), a wektor $\mathbf{Y}$ zawiera n składowych ($\mathbf{Y} \in \mathcal{R}^n$). Zadaniem sieci jest przyjęcie w trakcie procesu uczenia i zarejestrowanie (w postaci wag w_{ij}) zależności pomiędzy wejściem $\mathbf{X}$ i wyjściem $\mathbf{Y}$. W tym celu — jak zawsze — wykorzystywany jest ciąg uczący postaci

$$U = \{< X^{(1)}, Y^{(1)} >, < X^{(2)}, Y^{(2)} >, \ldots, < X^{(k)}, Y^{(k)} >, \quad \ldots, \\ \ldots, < X^{(N)}, Y^{(N)} >\}$$

w taki sposób, aby po ponownym pojawieniu się pewnego konkretnego wektora $\mathbf{X}^{(k)}$ (lub wektora *podobnego* do $\mathbf{X}^{(k)}$ — sieć jest *asocjacyjna*, więc jest w stanie *uogólniać* swoje doświadczenia) sieć odtwarzala na swoim wyjściu wektor $\mathbf{Y}^{(k)}$ zapamiętany jako skojarzony z danym $\mathbf{X}$.

Wektor $\mathbf{X}$ podawany jest na wejścia wszystkich neuronów wejściowej warstwy sieci, która jest połączona z warstwą wyjściową w taki sposób, że współczynniki wagowe tworzą macierz $\mathbf{W}$ o wymiarach $[m \times n]$. W wyniku przejścia sygnałów $x_j \in \mathbf{X}$ przez neurony wyjściowej warstwy powstaje wektor $\mathbf{Y}$, którego składowe otrzymuje się według znanej zależności:

$$y_i = \varphi \left(\sum_{j=1}^{m} w_{ij}\, x_j \right)$$

gdzie φ jest nielinową funkcją wiążącą wejście z wyjściem w pojedynczym neuronie. Zależność tę można zapisać w formie wektorowej

$$\boldsymbol{Y} = \boldsymbol{\Phi}\,(\boldsymbol{W}\ \boldsymbol{X})$$

gdzie: $\mathbf{W}$ jest macierzą wag wyjściowej warstwy sieci, $\mathbf{Y}$ jest wektorem wyjściowym drugiej warstwy sieci, a Φ jest wielowymiarowym nieliniowym odwzorowaniem

$$\Phi : \mathcal{R}^n \Rightarrow \mathcal{R}^n$$

stanowiącym formalny agregat odwzorowań realizowanych skalarnie przez funkcje φ, które zwykle przyjmowane są w postaci funkcji progowej albo sigmoidalnej funkcji logistycznej

$$\varphi\,(e) = \frac{1}{1 + e^{-\beta e}}$$

Przy przyjęciu funkcji progowej łatwiejsza jest analiza jakościowa działania sieci i łatwiej jest sobie wyobrazić jej zachowanie, natomiast funkcja logistyczna wykazuje swoje zalety przy próbach matematycznej analizy działania sieci. Przy realizacji praktycznej (np. w postaci układu elektronicznego lub programu symulacyjnego) można swobodnie wybrać taką formę nieliniowości, jaka jest wygodniejsza; zresztą możliwe jest tu płynne przechodzenie od funkcji logistycznej do funkcji progowej, ponieważ przy $\lambda \to \infty$ funkcja logistyczna przekształca się w funkcję progową. W dalszych rozważaniach opierać się będziemy na modelu funkcji progowej. Wynikiem takiego założenia jest fakt, że wszystkie sygnały w tej sieci są *bipolarne*, tzn.

$$\forall\, i \in [1,m]\ x_i \in \{-1,1\}$$

oraz

$$\forall\, j \in [1,n]\ y_j \in \{-1,1\}$$

Ma to dość istotny wpływ na zachowanie sieci, którego śledzenie jest dzięki temu łatwiejsze niż w przypadku sygnałów przyjmujących (jak typowo zakłada się dla innych sieci) wartości 0 i 1. Dodatkowym założeniem dotyczącym elementów wchodzących w skład sieci BAM jest przypisanie im własności „histerezy". Objawia się ona pewną „niechęcią" każdego neuronu do zmiany stanu. Przy założeniu, że funkcja wyjścia φ ma formę progową oznaczać to będzie, że przejście od wartości -1 do $+1$ lub na odwrót musi być każdorazowo wymuszone wyraźnie dodatnim lub ujemnym sygnałem wejściowym. Jeśli ważona suma sygnałów wejściowych, docierających do określonego neuronu, ma wartość 0, wówczas element ten **utrzymuje taki sam sygnał wyjściowy, jaki miał poprzednio**. Wprowadzając dyskretną skalę czasu t można to zapisać następująco:

$$y_i\,(t+1) = \begin{cases} +1 & \text{gdy} \quad \sum_{j=1}^{m} w_{ij}\, x_j(t) > 0 \\ -1 & \text{gdy} \quad \sum_{j=1}^{m} w_{ij}\, x_j(t) < 0 \end{cases}$$

natomiast przypadek $\sum_{j=1}^{m} w_{ij}\, x_j(t) = 0$ rozpatrywany jest osobno i wtedy

$$y_i\,(t+1) = y_i\,(t)$$

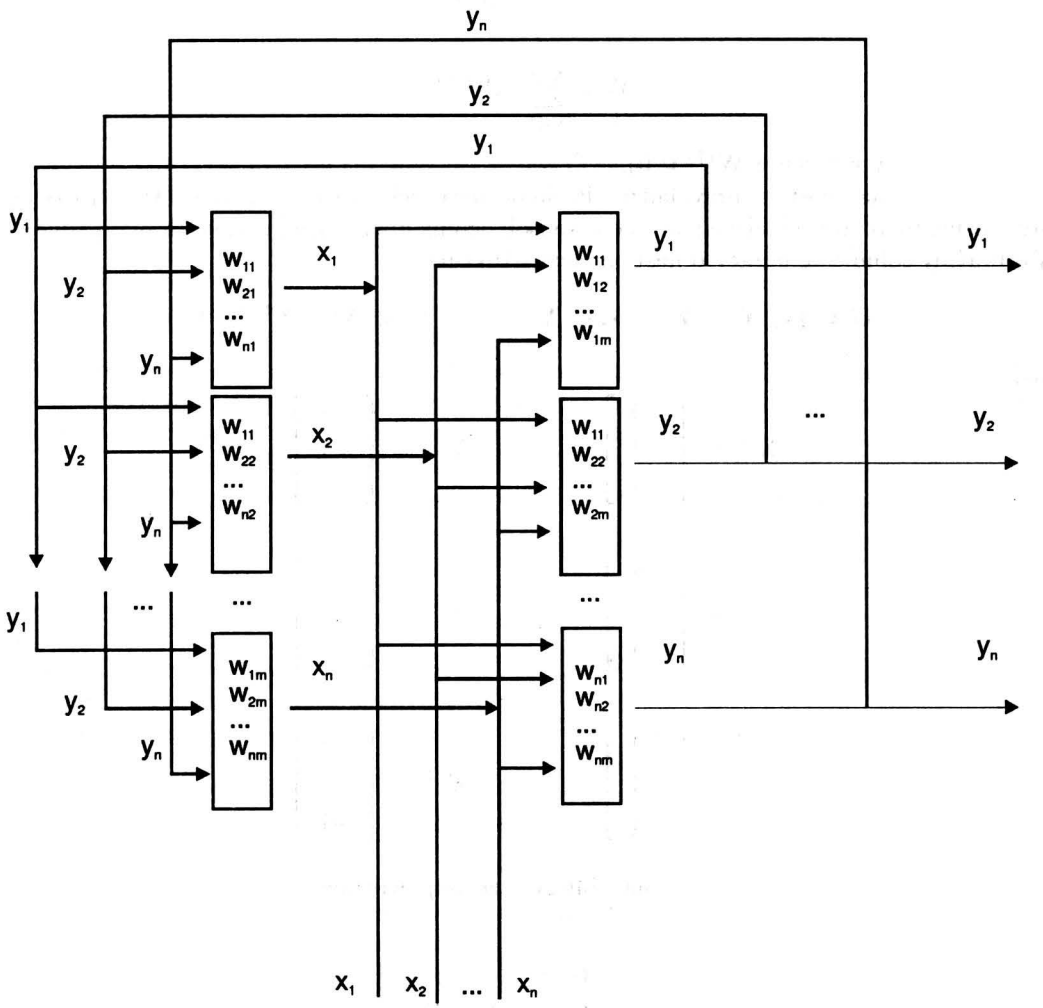
Wektor $\mathbf{Y}$ w sieci BAM zostaje skierowany na wejście pierwszej warstwy tworząc sprzężenie zwrotne, w którym wykorzystywana jest ta sama macierz $\mathbf{W}$, z tym, że oczywiście transponowana (ze względu na to, że w ogólnym przypadku wymiary wektorów $\mathbf{X}$ i $\mathbf{Y}$ mogą być różne), w związku z czym można zapisać kolejne równanie wektorowe

$$\boldsymbol{X} = \boldsymbol{\Phi}\,(\boldsymbol{W}^T\, \boldsymbol{Y})$$

Jak wynika z przedstawionego opisu, sieć BAM ma bardziej regularną budowę, niż zwykle sieci Hopfielda czy Hintona, a ponadto jest tworem czysto technicznym, trudno bowiem uznać za prawdopodobny biologicznie proces uczenia utrzymujący stale ścisłą zgodność wartości pewnych wag synaptycznych w ustalonych parach neuronów. Nie zmienia to oczywiście bezspornej praktycznej przydatności sieci typu BAM i dowodzi, że dziedzina sieci neuronowych, wywodząca się pierwotnie z prób naśladowania środkami techniki schematów struktur zaobserwowanych w systemach nerowowych, zyskała obecnie zdolność do tworzenia autonomicznych metod i całkowicie oryginalnych konstrukcji. Jest to przejaw dojrzałości tej techniki i namacalny przejaw jej rozwoju.

Dzięki obecności w sieci BAM sprzężenia zwrotnego możliwe są w niej wewnętrzne obiegi sygnałów, podobnie jak w sieci Hopfielda. Sieć po podaniu sygnału wejściowego **X** zostaje pobudzona i poszukuje stanu równowagi, który osiąga „przypomniawszy sobie" odpowiedni sygnał Y. Zachowanie sieci można prześledzić poczynając od procesu uczenia.

Schemat sieci BAM pokazano na rysunku.



8.4 Uczenie sieci BAM i przykład jej działania

Uczenie sieci BAM podlega prawu Hebba przy współczynniku $\eta = 1$, zatem po pokazaniu w ciągu uczącym pary sygnałów $< \mathbf{X}^{(k)}, \mathbf{Y}^{(k)} >$ macierz wag $\mathbf{W}$ zostaje zmodyfikowana o składnik będący macierzą korelacji $\mathbf{X}^{(k)}$ i $\mathbf{Y}^{(k)}$.

$$W^{(k+1)} = W^{(k)} + \Delta W^{(k)} = W^{(k)} + X^{(k)} Y^{(k)T}$$

Po przeprowadzeniu procesu uczenia od początku do końca mamy więc następujący stan pamięci sieci

$$W = \sum_{k=1}^{N} \Delta W^{(k)}$$

(oczywiście przy założeniu $\mathbf{W}^{(1)} = 0$).

Rozważmy na prostym przykładzie działanie sieci, żeby rozumieć, co się w niej dzieje, gdy zaczniemy rozważać jej zachowanie w sposób ogólny dla bardziej złożonych przypadków. Wyobraźmy sobie ciąg uczący o następującej strukturze:

$$U = \{< X^{(1)}, Y^{(1)} >, < X^{(2)}, Y^{(2)} >, < X^{(3)}, Y^{(3)} >\}$$

gdzie

$$\mathbf{X}^{(1)} = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix}, \qquad \mathbf{Y}^{(1)} = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}$$

$$\mathbf{X}^{(2)} = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}, \qquad \mathbf{Y}^{(2)} = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}$$

$$\mathbf{X}^{(3)} = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}, \qquad \mathbf{Y}^{(3)} = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix}$$

Łatwo (choć nieco uciążliwie) można obliczyć, że odpowiednie macierze korelacji $\Delta W^{(k)}$ mają w tym przypadku postać:

$$\Delta W^{(1)} = \begin{bmatrix} -1 & -1 & 1 \\ 1 & 1 & -1 \\ 1 & 1 & -1 \end{bmatrix}; \qquad \Delta W^{(2)} = \begin{bmatrix} 1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \end{bmatrix};$$

$$\Delta W^{(3)} = \begin{bmatrix} -1 & 1 & 1 \\ -1 & -1 & 1 \\ 1 & -1 & -1 \end{bmatrix}$$

Wynikowa macierz wag ma postać

$$W = \begin{bmatrix} -1 & -1 & 3 \\ -1 & 3 & -1 \\ 3 & -1 & -1 \end{bmatrix}$$

Mając do dyspozycji gotową wynikową macierz wag (czyli zawartość pamięci sieci BAM) możemy prześledzić, jak sieć działa podczas odtwarzania zapamiętanych informacji. Niech na wejściu sieci pojawi się sygnał

$$X = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix}$$

Porównanie z ciągiem uczącym ujawnia, że jest to sygnał $\mathbf{X}^{(1)}$, który sieć powinna pamiętać. Obliczmy sygnał $\mathbf{Y}$, jaki wygeneruje sieć na swoim wyjściu:

$$Y = \Phi(WX) = \Phi\left(\begin{bmatrix} -1 & -1 & 3 \\ -1 & 3 & -1 \\ 3 & -1 & -1 \end{bmatrix}\begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix}\right) = \Phi\left(\begin{bmatrix} -3 \\ -3 \\ 5 \end{bmatrix}\right) = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}$$

Wystarczy rzut oka na ciąg uczący, by się upewnić, że wynik jest poprawny. Otrzymany wektor wynikowy odpowiada dokładnie wektorowi $\mathbf{Y}^{(1)}$, który w takcie procesu uczenia był kojarzony z podanym na wejście sieci wektorem $\mathbf{X} = \mathbf{X}^{(1)}$.

Sieć poprawnie odtwarza skojarzone informacje w obu kierunkach. Rozważmy przykładowo zachowanie sieci po podaniu na jej wejście $\mathbf{Y}$ sygnału odpowiadającego $\mathbf{Y} = \mathbf{Y}^{(1)}$.

$$X = \Phi(W^T Y) = \Phi\left(\begin{bmatrix} -1 & -1 & 3 \\ -1 & 3 & -1 \\ 3 & -1 & -1 \end{bmatrix}\begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}\right) = \Phi\left(\begin{bmatrix} 5 \\ -3 \\ -3 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix}$$

Jak widać, sieć zachowała się poprawnie i odtworzyła na wyjściu $\mathbf{X}$ sygnał odpowiadający $\mathbf{X} = \mathbf{X}^{(1)}$. Podobnie można stwierdzić poprawne działanie sieci dla dowolnego z uprzednio zapamiętanych sygnałów, na przykład dla $\mathbf{X} = \mathbf{X}^{(2)}$ sieć poprawnie odtwarza $\mathbf{Y} = \mathbf{Y}^{(2)}$

$$Y = \Phi(WX) = \Phi\left(\begin{bmatrix} -1 & -1 & 3 \\ -1 & 3 & -1 \\ 3 & -1 & -1 \end{bmatrix}\begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}\right) = \Phi\left(\begin{bmatrix} -3 \\ 5 \\ -3 \end{bmatrix}\right) = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}$$

Podobnych przeliczeń można oczywiście wykonać dowolnie dużo, jest to jednak zbyteczne. Natomiast warto rozważyć zachowanie sieci w przypadku, kiedy podawany do sieci wektor $\mathbf{X}$ lub $\mathbf{Y}$ nie odpowiada idealnie wektorowi prezentowanemu w trakcie procesu uczenia.

8.5 Działanie sieci BAM przy braku zgodności ze wzorcem

W sytuacji, kiedy podany do sieci BAM sygnał $\mathbf{X}$ albo $\mathbf{Y}$ nie jest idelanie zgodny z żadnym zapamiętanym wzorcem, sieć poszukuje poprawnej odpowiedzi w sposób iteracyjny. W wyniku tego pojawia się pewien (dłuższy lub krótszy) proces przejściowy i po pewnym czasie

osiągany jest stan równowagi, odpowiadający — jak zawsze przy sieciach zawierających sprzężenia zwrotne — minimum funkcji „energii" sieci. Prześledźmy to dokładniej.

Watość funkcji „energii" związana z pojawieniem się w sieci BAM sygnałów $\mathbf{X}$ i $\mathbf{Y}$ wyraża się wzorem

$$\boldsymbol{E}(\boldsymbol{X},\boldsymbol{Y}) = -\boldsymbol{X}^T \boldsymbol{W}\boldsymbol{Y}$$

Ponieważ

$$W = \sum_{k=1}^{N} \Delta W^{(k)}$$

przeto

$$E(X,Y) = \sum_{k=1}^{N} E^{(k)}(X,Y)$$

gdzie

$$\boldsymbol{E}^{(k)}(\boldsymbol{X},\boldsymbol{Y}) = -\boldsymbol{X}^T \boldsymbol{\Delta W}^{(k)} \boldsymbol{Y}$$

co uwzględniając fakt, że

$$\boldsymbol{\Delta W}^{(k)} = \boldsymbol{X}^{(k)} \boldsymbol{Y}^{(k)T}$$

można przepisać w postaci

$$\boldsymbol{E}^{(k)}(\boldsymbol{X},\boldsymbol{Y}) = -\boldsymbol{X}^T (\boldsymbol{X}^{(k)} \boldsymbol{Y}^{(k)T}) \boldsymbol{Y}$$

albo jako

$$\boldsymbol{E}^{(k)}(\boldsymbol{X},\boldsymbol{Y}) = -(\boldsymbol{X}^T \boldsymbol{X}^{(k)})(\boldsymbol{Y}^{(k)T} \boldsymbol{Y})$$

Latwo zauważyć, że skladnik $\mathbf{E}^{(k)}(\mathbf{X},\mathbf{Y})$ ma tym mniejeszą wartość, im większe wartości mają iloczyny $(\mathbf{X}^T\mathbf{X}^{(k)})$ oraz $(\mathbf{Y}^{(k)T}\mathbf{Y})$, te zaś osiągają maksimum przy

$$\boldsymbol{X} = \boldsymbol{X}^{(k)}$$

oraz

$$\mathbf{Y} = \mathbf{Y}^{(k)}$$

Jest to teoretyczne uzasadnienie wykazanej już wyżej wlasności sieci polegającej na dążeniu do odpowiadania na bodźce $\mathbf{X}$ analogiczne z pokazanymi w ciągu uczącym $\mathbf{X}^{(k)}$ bodźcami $\mathbf{Y}$ analogicznymi do skojarzonych $\mathbf{Y}^{(k)}$. Jeśli jednak na wejściu sieci pojawi się sygnal $\mathbf{X} \neq \mathbf{X}^{(k)}$ dla wszystkich k, to wówczas w sieci rozpocznie się proces dynamiczny opisywany w kolejnych chwilach czasowych τ równaniami iteracyjnymi

$$\begin{aligned} \mathbf{Y}(\tau+1) &= \Phi(\mathbf{W}\,\mathbf{X}(\tau)) \\ \mathbf{X}(\tau+1) &= \Phi(\mathbf{W}\,\mathbf{Y}(\tau+1)) \end{aligned}$$

Przyjmując $\mathbf{X}(0) = \mathbf{X}$ otrzymujemy kolejno $\mathbf{Y}(1)$, $\mathbf{X}(1)$, $\mathbf{Y}(2)$, $\mathbf{X}(2), \ldots, \mathbf{Y}(i)$, $\mathbf{X}(i)$, $\mathbf{Y}(i+1)$, $\mathbf{X}(i+1)$). Rozważmy, jak się przy tym zmienia „energia" sieci. Przejście energii od wartości

$$E(i) = -\mathbf{X}(i)^T \mathbf{W}\,\mathbf{Y}(i)$$

do wartości

$$E(i+1) = -\mathbf{X}(i)^T \mathbf{W}\,\mathbf{Y}(i+1)$$

musi zmniejszać łączną energię, co łatwo uzasadnić faktem, że składowe wektorów $\mathbf{X}$ i $\mathbf{Y}$ są bipolarne, przeto zmiana wartości z $E(i)$ na $E(i+1)$ może zajść wyłącznie w wyniku **zmiany znaku** niektórych składowych wektora $\mathbf{Y}(i+1)$ w stosunku do $\mathbf{Y}(i)$. Ale z równania opisującego sieć

$$\mathbf{Y}(i+1) = \Phi\left(\mathbf{W}\,\mathbf{X}(i)\right)$$

wynika, że zmiany te mogły nastąpić wyłącznie zgodnie ze znakami odpowiednich składowych wektora $\mathbf{X}(i)^T\mathbf{W}$, co dowodzi, że energia sieci mogła w wyniku tych zmian jedynie zmaleć. Podobny argument można odnieść także do drugiego kroku iteracji

$$\mathbf{X}(i+1) = \Phi\left(\mathbf{W}\,\mathbf{Y}(i+1)\right)$$

Z tego wynika, że sieć błądząc od jednego stanu $< \mathbf{X}(i), \mathbf{Y}(i) >$ do kolejnego stanu $< \mathbf{X}(i+1), \mathbf{Y}(i+1) >$ porusza się **zawsze w kierunku wynikającym z malejącej energii**. Nie ulega wątpliwości, że taki proces może się zakończyć jedynie odnalezieniem lokalnego minimum funkcji energii, przy czym z wcześniejszych rozważań wynikało, że minima takie odpowiadają stanom wyznaczonym przez pary $< \mathbf{X}^{(k)}, \mathbf{Y}^{(k)} >$ pochodzące z prezentacji ciągu uczącego, zaś z faktu startowania procesu poszukiwań od podanej wartości $\mathbf{X}$ (lub $\mathbf{Y}$) stwarza szansę, że odnalezione minimum odpowiadać będzie najbliższej (na przykład w sensie normy euklidesowej) parze $< \mathbf{X}^{(k)}, \mathbf{Y}^{(k)} >$ w stosunku do startowego punktu $< \mathbf{X}, o >$ lub $< o, \mathbf{Y} >$. Niestety to ostatnie nie jest nigdy pewne — zdarza się, że sieć odnajdzie punkt równowagi zaskakująco odległy od początkowej wartości wektora $\mathbf{X}$ czy $\mathbf{Y}$. Badania nad prawami rządzącymi tymi zjawiskami są nadal w toku i brak jeszcze ostatecznych rozstrzygnięć. Optymistyczny akcent jest taki, że **na ogół** sieć odnajduje najbliższe asocjacje.

Interpretując omówione wyżej wyniki w kategoriach opisowych, można powiedzieć, że podanie sieci BAM pewnej informacji wejściowej $\mathbf{X}$ lub wyjściowej $\mathbf{Y}$ prowadzić będzie do odtworzenia przez sieć pewnej „kojarzącej się" pary zapamiętanych informacji: wejściowej i wyjściowej.

W sieci BAM, jak w każdym systemie ze sprzężeniem zwrotnym, mogą pojawić się oscylacje. To niekorzystne zjawisko może stawiać pod znakiem zapytania rzeczywistą użyteczność tych sieci. **Kosko** wykazał jednak [Kosk87], że wszystkie sieci BAM są bezwarunkowo stabilne bez względu na to, jaka jest macierz wag $\mathbf{W}$. Ta ważna własność wynika z faktu wykorzystania w strukturze BAM tej samej macierzy wag do połączenia pierwszej warstwy z drugą i drugiej warstwy z pierwszą.

Łatwo zauważyć, że sieć BAM sprowadza się do sieci Hopfielda w przypadku kiedy macierz $\mathbf{W}$ jest kwadratowa ($m = n$) i symetryczna, gdyż wtedy macierz $\mathbf{W} = \mathbf{W}^T$ może być uznana za macierz Hopfielda, a obie warstwy neuronów sieci BAM mogą być zebrane w jeden wspólny zbiór.

8.6 Pojemność pamięci sieci BAM

Sieć BAM jest pamięcią, więc jej działanie polega na rejestrowaniu informacji. Sensowne jest więc pytanie o pojemność tej pamięci czyli liczbę informacji L, jaka może być w niej zapamiętana. W literaturze pojawiają się różne oszacowania. **Kosko** [Kosk87] przyjmował, że pojemność pamięci sięga liczby neuronów w mniejszej z dwóch warstw sieci

$$L = \min\{n, m\}$$

Jest jednak dziś oczywiste, że ten wynik można osiągnąć jedynie przy bardzo specjalnym kodowaniu informacji w wektorach **X** i **Y**. **McEliece, Posner, Rodemich i Vankatesh** [McEl87] wykazali, że realistyczne jest oszacowanie

$$L = n/(4 \log_2 n)$$

gdzie przyjęto, że n jest liczbą neuronów w mniejszej warstwie. Warto zauważyć, jak bardzo różni się oszacowanie McEliece od optymizmu Kosko. Przy $n = 1024$ Kosko oczekuje możliwości zapamiętania ponad tysiąca informacji, a McEliece pozwala na zapamiętanie zaledwie 25!

Oczywiście są także i inne oszacowania. **Hecht-Nielsen** dowodził [Hech88], że indywidualnie dobierając progi wszystkich neuronów w sieci BAM można uzyskać w niej 2^n rozróżnialnych stanów, a co za tym idzie aż tyle jest potencjalnie możliwych do zapamiętania informacji. Oczywiście w praktyce nie można liczyć na taką wydajność, jednak dobierając wektory **X** i **Y** w taki sposób, by w każdym z nich było

$$l^+ = 4 + \log_2 n$$

składowych wynoszących $+1$, a pozostałe mające wartości -1, to wówczas można uzyskać pojemność pamięci

$$L = \frac{0,68\, n^2}{(\log_2 n + 4)^2}$$

większą od n. Przykładowo dla $n = 1024$ L wynosi aż 3637.

8.7 Odmiany sieci BAM

Obok omówionych wyżej sieci BAM w formie cyfrowej (tj. akceptujących jedynie wartości x_i i y_i wynoszące $+1$ albo -1) rozważane są sieci tego typu o elementach analogowych, z funkcją φ opisaną na przykład cytowaną wyżej sigmoidą logistyczną. Sieci takie okazują się także bardzo przydatne w adaptacyjnym przetwarzaniu sygnałów i budzą rosnące zainteresowanie badaczy zwłaszcza, że możliwa jest ich hardware'owa implementacja w układach VLSI oraz optoelektronicznych. Podobnie rozważane są sieci BAM o działaniu ciągłym (wyżej przytoczona dyskusja odnosiła się do dyskretnej skali czasu i angażowała pewne procesy iteracyjne w sieci). Sieci takie są możliwe do praktycznego zastosowania, ponieważ teoria Kosko zapewnia także ich stabilność.

Inną odmianą sieci BAM, dyskutowaną w literaturze [Wass89], jest sieć **adaptacyjna**. W sieci takiej dokonuje się permanentna powolna zmiana współczynników wagowych, zgodnie ze wzorem

$$w_{ij} = w_{ij} + \eta\, x_i\, y_j$$

Sieć taka może doskonalić swoje działanie w trakcie eksploatacji i nie wymaga oddzielnego procesu uczenia.

Jeszcze inna odmiana sieci BAM związana jest z wprowadzeniem do niej elementu **rywalizacji** (tylko jeden element w każdej warstwie neuronów ma sygnał wyjściowy wynoszący $+1$, pozostałe mają wymuszony sygnał wynoszący -1, nawet jeśli ważona suma ich wejść jest większa od założonego progu). Sieć tego typu może służyć do kojarzenia specjalnych typów wektorów binarnych — na przykład w diagnostyce medycznej.

Różnych odmian sieci BAM jest wiele. Jest ona szczególnie chętnie badana i analizowana przez inżynierów, ponieważ jej dzialanie szczególnie latwo daje się powiązać z konkretnymi zadaniami (na przyklad z zastosowaniami systemów ekspertowych). Niektórzy autorzy twierdzą, że sieć BAM stanowi najistotniejszy przyczynek do wykorzystania sieci neuronowych w bliskiej przyszlości, dlatego warto bylo ją tak szczegółowo tu zaprezentować.

Rozdział 9

Dynamika procesu uczenia sieci neuronowych

W poprzednich rozdzialach prezentowane byly rozmaite modele sieci neuronowych i różne związane z nimi techniki uczenia. Uczenie (rozważane tu chwilowo w odniesieniu do pojedynczego neuronu) generalnie polegalo na znajdowaniu nowych wartości wektora wag $\mathbf{W}'$ na podstawie poprzednich wartości tego wektora $\mathbf{W}$ oraz pewnych dodatkowych danych (wektora wejściowego $\mathbf{X}$, sygnalu wyjściowego y, wielkości zadanej z itp.). W wyniku procesu uczenia wektor wag $\mathbf{W}$ zmienial się w trakcie procesu uczenia, co zaznaczane bylo przez dopisywanie do tego wektora numeru kroku procesu uczenia j, najczęściej w postaci $\mathbf{W}^{(j)}$. Nasza dotychczasowa troska w trakcie procesu uczenia wiązala się głównie z efektem końcowym — czy neuron nauczy się wymaganej funkcji $y = f(\mathbf{X})$ czy nie. Teraz natomiast zainteresujemy się bliżej samym procesem uczenia i dynamiką zmian wektora wag $\mathbf{W}$ w trakcie tego procesu. Wygodniej będzie jednak rozpatrywać zmiany wag jako proces *ciągły*, wprowadzając wektor wag jako funkcję czasu $\mathbf{W}(t)$, a regulę uczenia jako równanie różniczkowe wiążące zmiany tego wektora z czynnikami decydującymi o procesie uczenia. Wzorując się na podejściu Kohonena [Koho89] ogólną postać tego równania można zapisać w formie

$$\frac{d\boldsymbol{W}}{dt} = \phi(.)\ \boldsymbol{X} - \gamma(.)\ \boldsymbol{W}$$

gdzie $\phi(.)$ i $\gamma(.)$ są pewnymi (być może nieliniowymi) skalarnymi funkcjami $\mathbf{X}$, $\mathbf{W}$ oraz y. Skladnik $\phi(.)\ \mathbf{X}$ odpowiedzialny jest za proces nabywania nowych doświadczeń przez sieć; powoduje on w ogólnym przypadku calkowanie sygnalów wejściowych i przyczynia się do wytworzenia wewnętrznej reprezentacji nabywanych przez neuron umiejętności. Drugi skladnik odpowiada za proces zapominania starych i nie aktualnych umiejętności, dzięki czemu mimo stalego doplywu nowych informacji $\mathbf{X}(t)$ nie dochodzi do przepelnienia pamięci.

W podanych niżej rozważaniach ograniczymy się do zadań, w których $\phi = \phi(y)$ i $\gamma = \gamma(y)$ analizując poszczególne przypadki funkcji o różnym stopniu nieliniowości.

Przypadek 1. Funkcje stale: $\phi = \alpha$ i $\gamma = \beta$

$$\frac{d\boldsymbol{W}}{dt} = \alpha\ \boldsymbol{X} - \beta\ \boldsymbol{W}$$

Rozwiązanie ma ogólną postać

$$\mathbf{W}(t) = e^{-\beta t}\left[\mathbf{W}(0) + \alpha \int_0^t e^{\beta\tau}\, \mathbf{X}(\tau)\, d\tau\right]$$

i można mu przypisać prostą interpretację: Uczenie prowadzi w tym wypadku jedynie do wyznaczania ruchomej średniej (ważonej wykładniczą funkcją czasu) wejściowego wektora $\mathbf{X}(t)$. Początkowa wartość wektora $\mathbf{W}(0)$ jest szybko zapominana.

Przypadek 2. Jedna funkcja jest liniowa, a druga stała: $\phi = \alpha\, y$ i $\gamma = \beta$

$$\frac{dW}{dt} = \alpha\, y\, X - \beta\, W$$

Jest to — jak pisze Kohonen — pierwszy nietrywialny model procesu uczenia najniższego rzędu. W tym wypadku sygnał wyjściowy neuronu y ingeruje w proces uczenia w najprostszy z możliwych sposobów, wywodzący się z klasycznych prac Hebba [Hebb49].

Zakładając, że neuron jest typu ADALINE (z liniową funkcją przejścia φ) mamy oczywiście $y = \mathbf{W}^T\mathbf{X}$, a zatem

$$\frac{dW}{dt} = (\alpha\, \mathbf{X}\, \mathbf{X}^T - \beta\, \mathbf{I})\ \mathbf{W}$$

gdzie $\mathbf{I}$ jest macierzą jednostkową o rozmiarach $[n \times n]$. Równanie to można zapisać w wygodniejszej postaci

$$\frac{dW}{dt} = -\beta\ (\mathbf{I} - \lambda\, \mathbf{X}\, \mathbf{X}^T)\ \mathbf{W}$$

gdzie $\lambda = \alpha/\beta$. Właściwości tego równania łatwiej będzie prześledzić, jeśli wprowadzi się dyskretną skalę czasu. Wówczas kolejne wartości wektora $\mathbf{W}(t)$ (gdzie $t = 0, 1, 2, \ldots$) można wyznaczać z iteracyjnego równania

$$\mathbf{W}(t+1) = \left[(1-\beta)\, \mathbf{I} + \alpha\, \mathbf{X}(t)\, \mathbf{X}^T(t)\right]\ \mathbf{W}(t)$$

Oznaczając występujący przy $\mathbf{W}(t)$ zależny od czasu (numeru kroku t) macierzowy mnożnik w tym równaniu przez $\mathbf{P}(t)$ otrzymujemy proste w formie równanie dynamiki procesu uczenia:

$$\mathbf{W}(t+1) = \mathbf{P}(t)\, \mathbf{W}(t)$$

gdzie macierz $\mathbf{P}(t)$ wyznaczana jest za pomocą zależności:

$$\mathbf{P}(t) = (1-\beta)\, \mathbf{I} + \alpha\, \mathbf{X}(t)\, \mathbf{X}^T(t)$$

Rozwiązanie opisanego równania daje dynamikę procesu uczenia w formie:

$$\mathbf{W}(t+1) = \left[\prod_{k=0}^{t} \mathbf{P}(k)\right]\ \mathbf{W}(0)$$

Dość łatwo jest się zorientować, że rozwiązanie to w ogólnym przypadku ma dość niekorzystne właściwości: albo jest rozbieżne (wartości $\mathbf{W}(t)$ „eksplodują" i osiągają nieskończenie duże wartości), albo zbiega się do wektora zerowego. Tak więc rozważaną tu metodę uczenia wolno stosować jedynie do modelowania systemów o skończonym i relatywnie krótkim

czasie uczenia. Przy takim założeniu opisana metoda uczenia może być wygodna jako technika aproksymująca w uproszczeniu zachowanie systemów o znacznie większym stopniu złożoności, które charakteryzują się asymptotyczną stabilnością, ale są znacznie trudniejsze do analizy i kosztowniejsze przy symulacji. W szczególności przy zaniedbaniu efektu „zapominania" (tzn. przy założeniu $\beta = 0$) rozwiązanie równania uczenia może być aproksymowane za pomocą wzoru:

$$\boldsymbol{W}(t) = \left[\boldsymbol{I} + \alpha \int_0^t \boldsymbol{X}(\tau)\, \boldsymbol{X}^T(\tau)\, d\,\tau\right] \boldsymbol{W}(0)$$

co oznacza, że wartości współczynników wagowych są w tym wypadku zależne tylko od wartości macierzy korelacji wejściowych sygnałów $\mathbf{X}(t)$. Jest to ważny fakt, pozwalający często na wygodną intepretację niektórych aspektów działania większych i bardziej złożonych sieci, a ponadto mający duże znaczenie praktyczne — wystarczy uświadomić sobie jak często w różnych kontekstach wykorzystywane są macierze korelacji w przetwarzaniu sygnałów.

Oczywiście podane wyżej przybliżone rozwiązanie $\mathbf{W}(t)$ jest nadal rozbieżne

$$(\lim_{t \Rightarrow \infty} \|\mathbf{W}(t)\| \Rightarrow \infty),$$

jednak można wykazać, że $\mathbf{W}(t)$ dąży do wektora własnego macierzy korelacji sygnałów $\mathbf{X}(t)$ odpowiadającego największej wartości własnej, w związku z czym może być wykorzystany dla dowolnego konkretnego $t < \infty$, dla określenia np. składowych kanonicznych sygnału. Szczególnie interesujący przypadek specjalny pojawia się, gdy założymy wartość $\alpha < 0$ i przyjmiemy, że wektor $\mathbf{X}(t)$ przyjmuje wartości z pewnego ograniczonego zbioru, gdyż wtedy $\mathbf{W}(t)$ jest zbieżny do pewnego skończonego, niezerowego wektora mającego interpretację ortogonalnego rzutowania, co będzie dalej szerzej omówione.

Przypadek 2A. Opisany wyżej przypadek liniowej funkcji Φ i stałej funkcji γ ma pewien szczególnie ważny podprzypadek, uzyskiwany przy założeniu, że $\alpha < 0$ i $\beta = 0$. Dla silniejszego zaznaczenia specyfiki tego zadania w podanych dalej wzorach przyjmować będziemy $\alpha > 0$ i zaznaczać będziemy ujemną wartość odpowiedniego składnika poprzez jawnie wpisywany znak (minus). Tego rodzaju proces uczenia opisują więc równania:

$$\begin{aligned} \frac{d\boldsymbol{W}^T}{dt} &= -\alpha\, \boldsymbol{y}\, \boldsymbol{X}^T \\ \boldsymbol{y} &= \boldsymbol{W}^T \boldsymbol{X} \end{aligned}$$

Zachowanie takiego neuronu łatwo można wydedukować na podstawie rozważenia rozwiązania $\mathbf{W}(t)$ otrzymanego dla $\mathbf{X}(t) = \text{const}$ dla $t > t_0$. Rozwiązanie to ma postać:

$$\boldsymbol{W}^T(t) = \boldsymbol{W}^T(t_0)\ \left[\boldsymbol{I} - \omega(t)\, \boldsymbol{X}\, \boldsymbol{X}^T\right]$$

gdzie

$$\omega(t) = \|\boldsymbol{X}\|^{-2}\ \left(1 - e^{-\alpha \|X\|^2 (t - t_0)}\right)$$

jak łatwo zauważyć, że $0 \leq \omega(t) \leq \|\mathbf{X}\|^{-2}$ i przy niezmiennym sygnale wejściowym „ślad pamięciowy" wejściowych wrażeń w miarę upływu czasu systematycznie się zaciera. Takie

zjawisko jest znane w biologii pod nazwą „habituacji"[1], a w teorii sieci neuronowych jest znane pod nazwą „wykrywacza nowości" *(novelity detector)*.

Przypadek 3. Tutaj także jedna funkcja jest liniowa, a druga stała, tylko odwrotnie, niż w poprzednio omówionym przypadku: $\phi = \alpha \quad \gamma = \beta\, y$. Wówczas

$$\frac{dW}{dt} = \alpha\, X - \beta\, y\, W$$

Na pozór równanie to jest bardzo podobne do wyżej omówionego, jednak wstawienie liniowego równania opisującego funkcjonowanie neuronu ($y = \mathbf{W}^T\, \mathbf{X}$) ujawnia natychmiast rzeczywistą złożoność rozważanego tu problemu, ponieważ równanie dynamiki uczenia

$$\frac{dW}{dt} = (\alpha\, I - \beta\, W\, W^T)\; X$$

jest w tym wypadku **nieliniowe** ze względu na $\mathbf{W}$, a więc nie może być w ogólnym przypadku rozwiązane analitycznie. Istotnie, podane wyżej równanie jest szczególną postacią **równania Riccatiego**, którego całka nie jest znana w postaci analitycznej. Naturalnie w konkretnym przypadku zawsze pozostaje do dyspozycji rozwiązanie numeryczne, jednak jego przydatność jest bardzo ograniczona. Możliwe jest jedynie uzyskanie pewnych ogólnych informacji o wartościach $\mathbf{W}(t)$ przy dość oczywistych i możliwych do spełnienia w praktyce założeniach upraszczających. Przykładowo wymnażając obie strony podanego wyżej równania przez $2\,\mathbf{W}^T$ otrzymujemy równanie

$$2\,\mathbf{W}^T \frac{dW}{dt} = 2\,\mathbf{W}^T\, (\alpha\, \mathbf{I} - \beta\, \mathbf{W}\, \mathbf{W}^T)\; \mathbf{X}$$

Równanie to jest już równaniem *skalarnym* (nie wektorowym), a jego jedyną niewiadomą jest kwadrat modułu wektora $\mathbf{W}$. Zapisując to w sposób jawny mamy:

$$\frac{d}{dt}\,(\|\mathbf{W}\|^2) = 2\,y\,(\alpha - \beta\,\|\mathbf{W}\|^2)$$

Można udowodnić, że rozwiązanie tego równania $\|\mathbf{W}\|^2$ dla $y > 0$ jest zbieżne do pewnej ustalonej wartości $\|\mathbf{W}^*\|^2 = \frac{\alpha}{\beta}$, co można interpretować w ten sposób, że długość wektora $\|\mathbf{W}\|^2$ nie ulega w trakcie uczenia istotnym zmianom, natomiast istota uczenia polega na tym, że wektor $\mathbf{W}$ jest *obracany* w taki sposób, aby dążył do uzgodnienia swego kierunku z kierunkiem wektora $\mathbf{X}$.

Podane wyżej rozważania wydają się być problematyczne, ponieważ w ogólnym przypadku trudno zagwarantować spełnienie warunku $y > 0$ dla dowolnego $\mathbf{X}$. Rozwiązaniem może tu być jednak uwzględnienie wpływu nieliniowej funkcji $\varphi(e)$, charakteryzującej się z reguły „odcinaniem" ujemnych wartości sygnałów y.

Zagadnienie obrotu wektora $\mathbf{W}$ w celu uzgodnienia jego położenia z kierunkiem wektora $\mathbf{X}$ (a dokładniej — z kierunkiem **oczekiwanego (średniego)** położenia wektora $\mathbf{X}$) rozważymy jako problem statystyczny. Wprowadźmy pojęcie **warunkowej wartości oczekiwanej**:

$$\mathbf{E}\{\cdot\,|\mathbf{W}\}$$

[1] Habituacja jest procesem stopniowego przywyczajania się systemu nerwowego do niezmiennego zestawu bodźców zmysłowych. W wyniku habituacji takie niezmienne bodźce po pewnym czasie przestają być świadomie dostrzegane.

oznaczającej uśrednioną po zbiorze wszystkich możliwych realizacji procesu uczenia wartość obiektu · (konkretna interpretacja tego obiektu będzie dalej określona). Wartość oczekiwana jest warunkowa (co zapisano z użyciem symbolu $|\mathbf{W}$). Na przykład można zapisać, że

$$\mathbf{E}\{\mathbf{X} \mid \mathbf{W}\} = \bar{\mathbf{X}}$$

gdzie $\bar{\mathbf{X}}$ oznacza wartość średnią $\mathbf{X}$, ponieważ $\mathbf{X}$ jest statystycznie niezależne od $\mathbf{W}$.

Rozważmy teraz zmienność kąta Θ pomiędzy wektorami $\bar{\mathbf{X}}$ i $\mathbf{W}$ w toku procesu uczenia. Ponieważ z definicji

$$\cos\Theta = \frac{\bar{\mathbf{X}}^T \mathbf{W}}{\|\bar{\mathbf{X}}\| \, \|\mathbf{W}\|}$$

przeto

$$\mathbf{E}\left\{d(\cos\Theta)/dt \mid \mathbf{W}\right\} = \mathbf{E}\left\{\frac{d}{dt}\left(\frac{\bar{\mathbf{X}}^T \mathbf{W}}{\|\bar{\mathbf{X}}\| \, \|\mathbf{W}\|}\right) \mid \mathbf{W}\right\}$$

Rozpisując dalej tę równość otrzymujemy:

$$\mathbf{E}\left\{\frac{d}{dt}\left(\frac{\bar{\mathbf{X}}^T \mathbf{W}}{\|\bar{\mathbf{X}}\| \, \|\mathbf{W}\|}\right) \mid \mathbf{W}\right\} =$$

$$= \mathbf{E}\left\{\frac{d(\bar{\mathbf{X}}^T \mathbf{W})/dt}{\|\bar{\mathbf{X}}\| \, \|\mathbf{W}\|} - \frac{(\bar{\mathbf{X}}^T \mathbf{W})\, d(\|\mathbf{W}\|)/dt}{\|\bar{\mathbf{X}}\| \, \|\mathbf{W}\|^2} \mid \mathbf{W}\right\}$$

Składnik $\frac{d(\bar{X}^T W)/dt}{\|\bar{X}\| \, \|W\|}$ może być obliczony przez pomnożenie obu stron równania $\frac{dW}{dt} = (\alpha I - \beta W W^T)X$ przez czynnik $\frac{\bar{X}^T}{\|\bar{X}\| \, \|W\|}$. Podczas obliczania składnika $\frac{(\bar{X}^T W)\, d(\|W\|)/dt}{\|\bar{X}\| \, \|W\|^2} \mid W$ można wziąć pod uwagę równość $\frac{d(\|W\|^2)}{dt} = 2\|W\| \frac{d(\|W\|)}{dt}$. Po uwzględnieniu tych tożsamości uzyskuje się zależność:

$$\mathbf{E}\left\{d(\cos\Theta)/dt \mid \mathbf{W}\right\} = \frac{\alpha \|\bar{\mathbf{X}}\|^2 - \beta (\bar{\mathbf{X}}^T \mathbf{W})^2}{\|\bar{\mathbf{X}}\| \, \|\mathbf{W}\|} - \frac{(\bar{\mathbf{X}}^T \mathbf{W})^2 (\alpha - \beta \|\mathbf{W}\|)^2}{\|\bar{\mathbf{X}}\| \, \|\mathbf{W}\|^3}$$

która po uproszczeniu przyjmuje postać:

$$\mathbf{E}\left\{d(\cos\Theta)/dt \mid \mathbf{W}\right\} = -\frac{\alpha \|\bar{\mathbf{X}}\|}{\|\mathbf{W}\|} \sin\Theta$$

Łatwo zauważyć, że dla niezerowych wartości $\bar{\mathbf{X}}$ „uśredniany" w toku procesu uczenia kierunek wektora $\mathbf{W}$ zmierza monotonicznie do kierunku wyznaczonego przez $\bar{\mathbf{X}}$. Ponieważ wyżej wykazano, że długość wektora $\mathbf{W}$ jest zbieżna do pewnej ustalonej wartości $\|\mathbf{W}^*\|^2 = \frac{\alpha}{\beta}$, zatem ogólnie można wykazać, że $\mathbf{W}$ w trakcie procesu uczenia zmierza do wektora $\mathbf{W}^* = \mathbf{W}(\infty)$, przy czym

$$\mathbf{W}^* = \frac{\sqrt{\alpha}}{\sqrt{\beta}\,\|\bar{\mathbf{X}}\|} \bar{\mathbf{X}}$$

jest w istocie wektorem $\bar{\mathbf{X}}$ o znormalizowanej długości $\sqrt{\frac{\alpha}{\beta}}$.

Do tego samego wniosku można dojść także na innej drodze. Rozważmy tak zwany **punkt stały** równania dynamiki uczenia, to znaczy taką wartość $\mathbf{W}^*$, dla której rozwiązanie równania

$$\frac{dW}{dt} = (\alpha\, \mathbf{I} - \beta\, \mathbf{W}\, \mathbf{W}^T)\, \mathbf{X}$$

przestaje się zmieniać, to znaczy $dW/dt = 0$. Rozważmy ten warunek przy podstawieniu $\mathbf{X} = \bar{\mathbf{X}}$ oraz $\mathbf{W} = \bar{\mathbf{W}}$. Wówczas

$$(\alpha\, \mathbf{I} - \beta\, \mathbf{W}^*\, \mathbf{W}^{*T})\, \bar{\mathbf{X}} = 0$$

Spróbujmy znaleźć rozwiązanie w postaci $\mathbf{W}^* = \rho \bar{\mathbf{X}}$, gdzie ρ jest poszukiwaną stałą. Po podstawieniu tej „odgadniętej" formuły otrzymuje się równanie

$$\alpha\, \bar{\mathbf{X}} - \beta\, \rho\, \bar{\mathbf{X}}\, \rho (\bar{\mathbf{X}}^T\, \bar{\mathbf{X}}) = 0$$

a z niego

$$\alpha\, \bar{\mathbf{X}} = \beta\, \rho^2\, ||\bar{\mathbf{X}}||\, \bar{\mathbf{X}}$$

i dalej

$$\rho = \frac{\sqrt{\alpha}}{\sqrt{\beta}\, ||\bar{\mathbf{X}}||}$$

a zatem

$$\mathbf{W}^* = \frac{\sqrt{\alpha}}{\sqrt{\beta}\, ||\bar{\mathbf{X}}||}\, \bar{\mathbf{X}}$$

Przypadek 4. Zagadnienie uczenia jest dwuliniowe, gdyż obie rozważane funkcje są liniowo zależne od y : $\phi = \alpha\, y$, $\gamma = \beta\, y$. Wówczas

$$\frac{dW}{dt} = \alpha\, y\, X - \beta\, y\, W$$

albo — uwzględniając równanie opisujące funkcjonowanie neuronu

$$\frac{dW}{dt} = \alpha\, X\, X^T\, W - \beta\, W\, W^T\, X$$

Wprowadzając oznaczenia:

$$\mathbf{E}\{\, \mathbf{X} \,|\, \mathbf{W} \,\} = \bar{\mathbf{X}}$$

oraz

$$\mathbf{E}\{\, \mathbf{X}\, \mathbf{X}^T \,|\, \mathbf{W} \,\} = \mathbf{C}_{\times\times}$$

gdzie $\mathbf{C}_{\times\times}$ jest **macierzą kowariancji** składowych wektora $\mathbf{X}$, otrzymujemy równanie:

$$\frac{dW}{dt} = \alpha\, \mathbf{C}_{\times\times}\, \mathbf{W} - \beta\, (\bar{\mathbf{X}}^T \mathbf{W})\, \mathbf{W}$$

łatwo rozpoznawalne jako równanie Bernouliego drugiego stopnia. Ustalenie zachowania rozwiązań tego równania nie jest tak łatwe, jak w poprzednio rozważanym przypadku: w szczególności poszukiwanie punktu stałego $\mathbf{W}^*$ za pomocą warunku $\frac{dW}{dt} = 0$ prowadzi do trywialnego rozwiązania $\mathbf{W}^* = 0$, całkowicie nieprzydatnego w naszych rozważaniach. Łatwo można jednak wykazać, że rozwiązaniem (określającym punkt stały równania) są

także **wektory własne macierzy kowariancji** $\mathbf{C}_{\times\times}$. Istotnie, spróbujmy szukać punktu stałego $\mathbf{W}^*$ w postaci $\mathbf{W}^* = \rho \mathbf{C}_i$, gdzie $\mathbf{C}_i$ jest wektorem własnym macierzy $\mathbf{C}_{\times\times}$ odpowiadającym wartości własnej λ_i:

$$\mathbf{C}_{\times\times}\, \mathbf{C}_\mathbf{i} = \lambda_\mathbf{i}\, \mathbf{C}_\mathbf{i}$$

wówczas równanie $\frac{dW}{dt} = 0$ prowadzi do zależności

$$\rho\, \alpha\, \lambda_i\, \mathbf{C}_i - \rho^2\, \beta\, (\bar{\mathbf{X}}^T\, \mathbf{C}_i)\, \mathbf{C}_i = 0$$

której rozwiązanie ma formę

$$\rho = \frac{\alpha\, \lambda_i}{\beta\, (\bar{\mathbf{X}}^T\, \mathbf{C}_i)}$$

i ostatecznie

$$\mathbf{W}^* = \frac{\alpha\, \lambda_i}{\beta\, (\bar{\mathbf{X}}^T\, \mathbf{C}_i)}\, \mathbf{C}_i$$

Podane wyżej rozwiązania (dla $i = 1, 2, \ldots, n$) mogą być stabilne lub niestabilne. Można się o tym przekonać rozważając zmienność kąta Θ pomiędzy wektorem $\mathbf{W}^*$ i $\mathbf{C}_i$.

$$E\left\{\frac{d}{dt}\left(\frac{C_i^T\, W}{\|C_i\|\, \|W\|}\right) | W\right\} =$$

$$= E\left\{\frac{d\,(C_i^T\, W)/dt}{\|C_i\|\, \|W\|} - \frac{(C_i^T\, W)\, d\,(\|W\|)/dt}{\|C_i\|\, \|W\|^2} | W\right\}$$

Po uwzględnieniu zależności $\mathbf{C}_i^T\, \mathbf{C}_{\times\times} = \lambda_i\, \mathbf{C}_i^T$ i po przekształceniach otrzymujemy zależność:

$$E\left\{\frac{d}{dt}\left(\frac{C_i^T\, W}{\|C_i\|\, \|W\|}\right) | W\right\} = \alpha \cos\Theta \left(\lambda_i - \frac{W^T\, C_{\times\times}\, W}{\|W\|^2}\right)$$

Twierdzenie Rayleigha głosi, że dla dowolnej macierzy $\mathbf{A}_{\times\times}$ i dowolnego wektora $\mathbf{V}$ spełniona jest nierówność

$$\frac{\boldsymbol{V}^T\, \boldsymbol{A}_{\times\times}\, \boldsymbol{V}}{\|\boldsymbol{V}\|^2} \leq \boldsymbol{\lambda}_{\max}$$

gdzie $\lambda_{\max}$ jest największą wartością własną macierzy $\mathbf{A}$. Stosując to twierdzenie do macierzy $\mathbf{C}_{\times\times}$ łatwo można stwierdzić, że rozwiązania opisujące przebieg procesu uczenia $\mathbf{W}(t)$ dążyć będzie do wartości $\mathbf{W}^* = \rho \mathbf{C}_{\max}$, gdzie $\mathbf{C}_{\max}$ jest wektorem własnym odpowiadającym wartości własnej $\lambda_{\max}$. Warunkiem jest jednak to, by w każdym momencie t zachodziła zależność $\mathbf{C}_{\max}^T\, \mathbf{W}(t) > 0$. Biorąc pod uwagę fakt, że wektor $\mathbf{C}_{\max}$ jest oczywiście a'priori nieznany — trudno **zagwarantować** spełnienie tego warunku, przy czym podstawowa trudność pojawia się przy ustalaniu punktu starowego dla procesu uczenia $\mathbf{W}(0)$, ponieważ oczywiście trzeba zapewnić spełnienie warunku $\mathbf{C}_{\max}^T\, \mathbf{W}(0) > 0$. Brak spełnienia wyżej sformułowanych warunków prowadzi zwykle od procesu uczenia, który jest niestabilny (rozbieżny do nieskończoności) albo zbieżny do $\mathbf{W}^* = 0$.

Przypadek 5. Zagadnienie uczenia jest z założenia nieliniowe, gdyż tylko jedna funkcja jest liniowo zależna od y : $\phi = \alpha y$, natomiast druga ma od początku formę nieliniową: $\gamma = \beta y^2$. Wówczas

$$\frac{dW}{dt} = \alpha\, y\, X - \beta\, y^2\, W$$

albo — uwzględniając równanie opisujące funkcjonowanie neuronu

$$\frac{dW}{dt} = \alpha X X^T W - \beta (W^T X W X^T) W$$

Wprowadzając jak poprzednio oznaczenie:

$$E\{X X^T \mid W\} = C_{xx}$$

otrzymujemy równanie:

$$\frac{dW}{dt} = \alpha C_{xx} W - \beta (W^T C_{xx} W) W$$

Ten model uczenia okazuje się — co może być zaskakujące — latwiejszy do analizy, niż wcześniej dyskutowane przypadki. Rozważmy najpierw długość wektora $\mathbf{W}$.

$$\mathbf{E}\left\{ d\|\mathbf{W}\|/dt \mid \mathbf{W} \right\} = 2\mathbf{W}^T \mathbf{C}_{xx} \mathbf{W} (\alpha - \beta \|\mathbf{W}\|^2)$$

Ponieważ $\mathbf{W}^T \mathbf{C}_{xx} \mathbf{W}$ jest skalarem i $\mathbf{W}^T \mathbf{C}_{xx} \mathbf{W} > 0$, zatem latwo dowieść, że $\|\mathbf{W}\|$ zmierza do wartości $\sqrt{\alpha/\beta}$, podobnie jak w przypadku 3.

Następnie rozważymy zmienność kąta Θ_i pomiędzy wektorem $\mathbf{W}^*$ i $\mathbf{C}_i$ (i-tym wektorem wlasnym macierzy $\mathbf{C}_{xx}$). Latwo wykazać, że

$$\mathbf{E}\left\{ d(\cos\Theta_i)/dt \mid \mathbf{W} \right\} = \alpha \cos\Theta_i \left(\lambda_i - \frac{\mathbf{W}^T \mathbf{C}_{xx} \mathbf{W}}{\|\mathbf{W}\|^2} \right)$$

czyli, że zachowanie wektora $\mathbf{W}$ jest tu analogiczne, jak w przypadku 4. W sumie rozwiązanie $\mathbf{W}(t)$ zmierza więc do punktu na powierzchni sfery o promieniu $\sqrt{\alpha/\beta}$, a położenie tego punktu wyznaczone jest przez wektor $\mathbf{C}_{max}$.

Opisane wyżej rozważania można teraz uogólnić. Rozważmy ponownie równanie dynamiki procesu uczenie w ogólnej postaci:

$$\frac{dW}{dt} = \phi(.) X - \gamma(.) W$$

Można sformulować następujące dwa ogólne **twierdzenia**:

Twierdzenie 1. Niech $\phi(.) = \alpha$ i $\gamma(.) = \gamma(y)$ a $y = W^T X$. Niech funkcja $\gamma(y)$ spelnia warunek, że istnieje wartość oczekiwana $\mathbf{E}\{\gamma(y) \mid \mathbf{W}\}$, a dla każdego t wektor $\mathbf{X}(t)$ niech będzie wektorem stochastycznym o stacjonarnych parametrach statystycznych, niezależnych od $\mathbf{W}$. Wówczas, jeśli równanie

$$\frac{dW}{dt} = E\{\alpha X - \gamma(y) W \mid W\}$$

ma niezerowe ograniczone rozwiązanie $\mathbf{W}^*$, to rozwiązanie to musi mieć **ten sam kierunek** co $\bar{\mathbf{X}}$ — wartość średnia $\mathbf{X}(t)$).

Twierdzenie 2. Przyjmijmy wszystkie założenia twierdzenia 1, a ponadto załóżmy, że $\mathbf{C}_{xx}$ jest macierzą kowariancji wektora $\mathbf{X}$. Wówczas jeśli równanie

$$\frac{dW}{dt} = E\{\alpha y X - \gamma(y) W \mid W\}$$

ma niezerowe ograniczone asymptotyczne rozwiązanie $\mathbf{W}^*$, to rozwiązanie to musi mieć **ten sam kierunek** co $\mathbf{C}_{max}$ — wektor własny macierzy $\mathbf{C}_{xx}$ odpowiadający największej wartości własnej λ_{max}.

Dowody obydwu twierdzeń znaleźć można w książce **Kohonena** [Koho89].

Rozdział 10

Przykłady konkretnych zastosowań sieci neuronowych

Dyskusja przedstawiona w poprzednich rozdzialach książki dotyczyla ogólnych właściwości sieci neuronowych i opierala się głównie na danych literaturowych. W tym ostatnim rozdziale książki zamiast próby podsumowania przedstawione zostaną dwa przyklady zastosowania sieci neuronowych, pochodzące z prac badawczych autora książki i jego wspólpracowników. Przyklady wybrano w taki sposób, by pokryć nimi możliwie szeroki zakres zastosowań sieci, będzie więc informacja o udanych próbach rozpoznawania mowy (dla sterowania robota dydaktycznego) oraz także przedstawione zostaną wyniki eksperymentów polegających na znajdowaniu optymalnego rozwiązania problemu komiwojażera, wykonywanych z użyciem zmodyfikowanej sieci Hopfielda. Prezentacja tych przykladów ma wykazać, że w polskich warunkach, mimo braku specjalizowanych neurokomputerów i mimo trudnego dostępu do komputerów o dużych mocach obliczeniowych, calkowicie możliwe jest uzyskiwanie sensownych zastosowań sieci neuronowych.

10.1 Rozpoznawanie wybranego zbioru wyrazów

10.1.1 Uwagi wstępne

Opisane w poprzednich rozdzialach sieci neuronowe stanowią narzędzia uniwersalne. Do konkretnego zadania, jakie rozwiązywano podczas realizacji opisywanych tu prac, trzeba bylo wiele z ogólnie wprowadzonych opisów i definicji skonkretyzować, ustalić i zdefiniować. Ten rozdzial, w odróżnieniu od wcześniejszych, prezentuje zatem wiadomości bardzo konkretne i szczegółowe, dotyczące tego, jak konkretnie uzyskano efekt w postaci efektywnego uczenia się sieci i jak konkretnie doprowadzono do tego, że sieć byla zdolna rozpoznawać wprowadzane do niej wypowiedzi.

10.1.2 Struktura sieci wybranej do badań

Do rozpoznawania poleceń glosowych dla robota użyto sieci neuropodobnej trójwarstwowej o 868 elementach. Warstwa pierwsza (wejściowa) posiadala zgodnie z założoną (agregacja)

ilością elementów widma częstotliwościowego sygnalu mowy 660 elementów, na warstwę wyjściową skladalo się ich 7 (ponieważ sieć miala rozpoznawać 7 rozkazów). Warstwa ukryta (środkowa) liczyla 200 elementów, ponieważ na tyle pozwalala rozszerzona do 4 Mb pamięć używanego do symulacji komputera (AT 386/387/33 MHz).

Zgodnie z zasadami rządzącymi doborem rodzaju sieci do tak skomplikowanego problemu uczenia wybrano sieć typu *back-propagation* z elementami podstawowymi typu sigmoidalnego. Kolejne warstwy łączono ze sobą metodą „każdy z każdym", warstwy ukrytą i wyjściową połączono dodatkowo z elementem typu „stala 1" (bias). Jak latwo policzyć w sumie powstalo

$$(n1 + 1)^* n2 + (n2 + 1)^* n3 = 133\ 607$$

połączeń, gdzie $n1$, $n2$ i $n3$ oznaczają odpowiednio ilości elementów w poszczególnych warstwach (1, 2 i 3). Przed rozpoczęciem procesu uczenia wagom tych połączeń zostaly przyporządkowane wartości pseudoprzypadkowe z przedzialu $[-0.1, 0.1]$.

10.1.3 Uczenie sieci

Głównym problemem, z jakimi borykają się wszyscy eksperymentatorzy, zajmujący się stosowaniem sieci neuronowych do konkretnych zadań jest odpowiednie poprowadzenie procesu uczenia. Dane literaturowe, ogólnie bardzo bogate na ten konkretny temat są niezwykle skąpe, a czasem także sprzeczne ze sobą. Ogólnie można tu wskazać na występowanie następujących problemów:

1. Dobór odpowiedniej wielkości zbioru uczącego, a następnie opracowanie metody mieszania danych podczas procesu uczenia.

2. Określenie wielkości współczynników uczenia $\eta 1$ (learning rate) i $\eta 2$ (momentum) por. rozdz. 4. oraz podjęcie decyzji o ich ewentualnej zmianie w trakcie procesu uczenia.

3. Zdefiniowanie czasu uczenia (w ilościach prezentacji wypowiedzi), a tym samym wypracowanie kryteriów pozwalających na ocenę procesu uczenia.

Podczas eksperymentów z siecią neuronową dopracowano się pewnych rozwiązań trzech powyższych grup problemowych, przy czym należy wyraźnie podkreślić, że rozwiązania te zależą bardzo ściśle od wielkości i struktury modelowanej sieci i nie mogą być bezkrytycznie przenoszone na inne, różne od prezentowanej, struktury. Równocześnie okazuje się, że przedstawiony wyżej podzial jest podzialem formalnym, a w rzeczywistości wszystkie trzy problemy są ze sobą ściśle sprzężone.

10.1.4 Zbiory uczące i zbiór testujący

Początkowo optymistycznie przyjęto, że do uczenia sieci wystarczy jeden zbiór uczący, w którym zapisano 70 wypowiedzi, czyli każda komenda dla robota zostala wypowiedziana 10 razy. Do testowania użyto zbioru skladającego się z 21 elementów: siedmiu rozkazów wypowiedzianych po 3 razy. Obydwa zbiory zostaly zarejestrowane przez tego samego mówcę. Czynnikiem utrudniającym naukę i rozpoznawanie byl fakt, że nagrania zbiorów nastąpily w różnym czasie i z różnym poziomem szumów nakladających się na sygnal mowy.

Po 200 prezentacjach zbioru uczącego (14 000 cykli uczenia) test wykazal, że sieć nauczyla się rozpoznawania zbioru uczącego (100%), natomiast zbiór testowy rozpoznawala jedynie

w 85% (18 wypowiedzi na 21). Wynik ten uznano za niezadawalający. Wobec tego, że sieć nie rokowała nadziei na „douczenie" (sprawdzano wyniki zbiorem testowym także na 100 i 140 kroku prezentacji zbioru uczącego) zdecydowano się na nagranie jeszcze jednego zbioru 70 wypowiedzi (10 razy każdy z rozkazów). W celach poznawczych nauczono sieć „od początku" rozpoznawania drugiego zbioru (100 prezentacji) i potraktowano oba zbiory uczące zamiennie jako testowe. Wyniki przedstawia tabela 1.

Tabela 1. Procentowy udział poprawnie rozpoznanych obiektów

Zbiór uczący	dług	ilość kroków	rozpoznawanie								
			zb. testowy			zbiór UCZ2			zbiór UCZ1		
			ile	roz.	%	ile	roz.	%	ile	roz.	%
UCZ1	10x7	200	21	18	85	70	53	75	uczący: 100 %		
UCZ2	10x7	100	21	19	90	uczący: 100 %			70	55	78

Z przebiegu procesu uczenia wynika, że oba zbiory uczące zostały prawidłowo zbudowane: sieć jest w stanie nauczyć się rozpoznawania prezentowanych wypowiedzi podczas około 100 prezentacji każdego z przyjętych zbiorów uczących. Niestety, nie można tego powiedzieć o wynikach rozpoznawania: zbiory traktowane jak testowe są rozpoznawane przez sieć w 75 - 90%. W tej sytuacji koniecznym staje się scalenie obu zbiorów uczących i ponowienie uczenia.

W każdym z procesów uczenia niebagatelną rolę odgrywa sposób prezentacji zbioru uczącego. Zwracają na to uwagę eksperymentatorzy uczący sieci, przy czym tradycyjnie nie ma tu gotowych metod postępowania gwarantujących sukces. Z procesu uczenia sieci wynika, że ogromną rolę odgrywa zarówno początkowe wymieszanie poszczególnych wypowiedzi jak i częstość mieszania ich w trakcie uczenia. Jeśli chodzi o tę ostatnią wielkość to (aby nie wydłużać i tak sporego czasu obliczeń) przyjęto, że kolejność elementów[1] w zbiorze uczącym jest zmieniana co 10 prezentacji całego zbioru (czyli co 700 pojedynczych cykli uczenia). Równocześnie zwracano uwagę na to, aby po kolejnym mieszaniu nie pojawiały się (przypadkowo) sekwencje jednakowych rozkazów. Z sytuacją taką ma się przeważnie do czynienia na początku procesu uczenia, kiedy to zbiór uczący jest uporządkowany, a dopiero potem poszczególne rozkazy — w miarę, gdy proces randomizacji postępuje — są szeregowane w kolejności przypadkowej.

10.1.5 Współczynniki uczenia

Jak wspomniano w rozdziale 4, współczynnik $\eta 1$ *(learning rate)* odpowiada za szybkość procesu uczenia (jest on mnożony przez propagowany wstecz błąd), $\eta 2$ natomiast (mnożony przez wielkość zmiany wag w poprzednim kroku) „wygładza" zbyt raptowne skoki wag połączeń. W literaturze podaje się ich „klasyczne" wartości jako odpowiednio: $\eta 1 = 0.9$ i $\eta 2 = 0.6$, przy czym zaznacza się, że w przypadku dużych sieci należy przyjmować wartości

[1] Elementami zbioru uczącego są pary złożone z wektorów sygnałów wejściowych (informacji podawanych na wejściową warstwę sieci) i wymaganych sygnałów wyjściowych (wzorcowych odpowiedzi sieci). W rozważanych zadaniach zbiór uczący tworzyły próbki rozpoznawanych wypowiedzi i informacje o ich prawidłowym znaczeniu.

mniejsze, lecz o podobnych proporcjach względem siebie. Podaje się również, że w trakcie uczenia wartości te powinny być zmniejszane w celu zapewnienia lepszej zbieżności uczenia.

Z drugiej strony inni autorzy [Chee90a] proponują startowanie wręcz od zerowych wartości współczynników $\eta 1$ i $\eta 2$ i stopniowe ich zwiększanie aż do osiągnięcia pewnej wartości ustalonej. W prezentowanych badaniach przyjęto rozwiązanie kompromisowe: startowanie od pewnych małych wartości współczynników i stopniowe ich zwiększanie. Tabela 2 prezentuje sposób zmian współczynników $\eta 1$ i $\eta 2$ w trakcie uczenia sieci różnymi zbiorami.

Tabela 2. Zmiany współczynników η_1 i η_2 w trakcie uczenia sieci

	η_1	0.15	0.3	0.6	0.9
	η_2	0.1	0.2	0.4	0.6
Zbiory UCZ1, UCZ2 (70 elem.)	l.kroków	do 3000	do 6000	do 9000	pow. 9000
	il. prez. zbioru	do ~ 42	do ~ 85	do ~ 128	pow. ~ 128
Suma a zbiorów UCZ1+ UCZ2 b (140 elem)	l.kroków	do 3000	do 6000	do 9000	pow. 9000
	il. prez. zbioru	do ~ 21	do ~42	do ~ 64	pow. ~ do 64
	l. kroków	do 6000	do 12000	do 18000	pow. 18000
	il. prez. zbioru	do ~ 42	do ~ 85	do ~ 128	pow. ~ 128

10.1.6 Czas uczenia

Czas uczenia najwygodniej określić przy pomocy ilości cykli prezentacji zbioru uczącego. Równocześnie powstaje problem oceny jakości uczenia po wykonaniu określonej ilości cykli. Podejście klasyczne, polegające na określaniu procentowym poprawności nauczenia lub rozpoznawania mówi niewiele o zmianach zachodzących przecież przez cały czas uczenia w sieci. Często zdarza się, że procent poprawnych odpowiedzi nie zmienia się lub zmienia się bardzo mało. W tej sytuacji dla sieci typu rozpoznającego, w której jeden element wyjściowy sygnalizuje rozpoznany obiekt lub klasę obiektów *(winner takes all)* proponuje się użycie jako mierników uczenia (rozpoznawania) znormalizowanych do przedziału [0, 1] wyjść ostatniej warstwy sieci.

W przypadku sieci z elementami sigmoidalnymi normalizacja dokonuje się automatycznie. Istotne są tu proste statystyczne funkcje wyjść, takie jak wartość maksymalna, minimalna i średnia, przy czym w zależności od potrzeb mogą one być liczone dla poszczególnych elementów poddawanych rozpoznawaniu lub dla całego ich zestawu.

Proces uczenia zakończono dla opisywanej sieci po 118 prezentacjach zbioru UCZ1+UCZ2 (140 elementów = 20x7 rozkazów). Wyniki uczenia (100% rozpoznania zbioru uczącego i 95% rozpoznania ciągu testowego) można uznać za zadowalający. Należy jednak stwierdzić, że proces uczenia sieci jest żmudny i czasochłonny. Przykładowo, obliczenia prowadzone na komputerze IBM AT 386/387 o częstotliwości zegara 33 MHz trwały — z przerwami — kilka dni dla każdego cyklu uczenia, a czas uczenia sieci jedną prezentacją zbioru 140-elementowego (UCZ1+UCZ2) wynosił około 10 minut!.

10.2 Rozwiązywanie problemu komiwojażera

10.2.1 Opis problemu i programu symulującego sieć

W opisanych badanich realizowano symulacyjnie sieć opisaną przez **Aiyera, Niraujana i Falliside** jako modyfikacja koncepcji zaproponowanej przez Hopfielda i Tanka. Zastosowanych w badaniach program opiera się o zmodyfikowany model sieci TSP (por. rozdz. 7). Istnieje w nim możliwość zmiany parametrów $A, A_1, C, D, i^b, \Delta t$ (z założenia $A = B$) oraz wyboru postaci funkcji f. Najczęściej przyjmowano funkcję w postaci

$$f(x) = \begin{cases} 0.1 + 0.1 \tanh(10(x + 0.4)) & \text{dla} \quad x \leq -0.4 \\ x + 0.5 & \text{dla} \quad -0.4 < x < 0.4 \\ 0.9 + 0.1 \tanh(10(x - 0.4)) & \text{dla} \quad x \geq 0.4 \end{cases}$$

Przy pomocy parametrów *Iter* i $Min\Delta V$ można określić kryteria stopu dla iterowania sieci TSP. *Iter* określa maksymalną liczbę iteracji, $Min\Delta V$ oznacza wartość, która powinna być wieksza od bezwzględnej wartości zmiany każdego z wyjść v_{Xi} w stanie uznanym za bliski ustalonego. *Sprawdz* określa okres sprawdzania kryterium stopu. *Proby* — przy jej pomocy można podać ile razy jest rozwiązywany ten sam problem, za każdym razem sieć startuje z innego stanu początkowego i generowanego losowo.

W obecnej wersji programu nie ma możliwości ręcznego wprowadznia odległości pomiędzy miastami, problemy podlegające rozwiązywaniu mogą być generowane tylko automatycznie. Rodzaj generowanego problemu określa się przez podanie N i stwierdzenie, że problem ma być *planarny* (miasta leżą na płaszczyźnie, w kwadracie jednostkowym, odległości pomiędzy nimi są długościami łączących je odcinków) lub *nieplanarny* (dla każdych dwóch miast jest generowana liczba losowa określająca ich odległość). Każdy nowo wygenerowany problem jest najpierw rozwiązywany za pomocą zwykłych (nie neuronowych) algorytmów optymalizacji: heurystyki FITSP i algorytmów zwanych w literaturze 2-optymalny i 3-optymalny. Następnie po otrzymaniu tych trzech rozwiązań, problem rozwiązywany jest za pomocą sieci TSP. Do podanej niżej statystyki są dołączane tylko rozwiązania tych problemów, dla których uzyskano co najmniej jedno rozwiązanie dopuszczalne siecią TSP. Statystyki są średnią arytmetyczną odpowiednio długości tras i czasów rozwiązywania. Liczona jest także średnia minimalna droga — jest to najkrótsza droga wyznaczona przez sieć TSP lub któryś z algorytmów klasycznych — dla danego problemu.

Celem badań było wykazanie podstawowych właściwości rozwiązań problemów optymalizacji uzyskiwanych za pomocą sieci neuronowych: jakości rozwiązania i jego niezawodności, wyrażającej się liczbą rozwiązań dopuszczalnych wśród rozwiązań proponowanych przez sieć. Dla zilustrowania pojawiających się tu problemów przedstawionio dalej i przeanalizowano dwa krańcowe przypadki. Warto dodać, że dla wybrania tych dwóch przypadków w rzeczywistości dokonano kilkuset symulacji rozważanych sieci i zebrano bardzo bogaty meterial badawczy, którego pełna prezentacja nie jest tu jednak oczywiście możliwa. Pierwszy z opisanych eksperymantów był w istocie powtórzeniem danych literaturowych, drugi stanowił oryginalny wynik uzyskany przez autora i współpracowników.

10.2.2 Wyniki eksperymentów

Przypadek 1. Modelowaną sieć charakteryzują następujące parametry (przyjęte wg. pracy

[Aiye90]):

N = 10,
typ = *nieplanarny*,
$A = 8$, $A_i = 7.75$, $C = 0.8$, $D = 1$,
$i^b = 8$, $\Delta t = 0.02$,
Proby=3, *MaxIter* = 3000, $Min\Delta V = 6e - 10$,
$\lambda_1 = -80$, $\lambda_2 = 0.5$, $\lambda_3 = -79.5$

Uzyskane wyniki podaje tabela 3, w której podano dla poszczególnych algorytmów uzyskany wynik (w postaci długości optymalnej drogi) oraz czas obliczeń. Ponieważ eksperymenty z siecią neuronową powtarzano wielokrotnie, wyniki podano w formie ststystyki podając wartość *średnią* i minimalną. U dołu tabeli 3 podano także poprawne rozpoznanie w postaci minimalnej drogi.

Tabela 3. Ocena rozwiązań uzyskanych za pomocą sieci neuronowych dla zadania TSP (Przypadek 1)

Algorytm	Droga	Czas
FITSP	3.6159	02s
2-opt	2.5789	0.23s
3-opt	2.3124	1.13s
Sieć śr.	2.2472	4min 9.15s
Sieć min.	2.2223	4min 31.60s
Minimal.	2.1919	

Przypadek 2. Modelowaną sieć charakteryzują następujące parametry:

N=10,
typ = *nieplanarny*,
$A = 8$, $A_1 = 7.75$, $C = 1.44$, $D = 8$,
$i^b = 14.4$, $\Delta t = 0.02$,
Proby = 3, *MaxIter* = 3000, $Min\Delta V = 6e - 10$,
$\lambda_1 = -144$, $\lambda_2 = 0.5$, $\lambda_3 = -79.5$

Wyniki optymalizacji są następujące:

Tabela 4. Ocena rozwiązań uzyskanych za pomocą sieci neuronowych dla zadania TSP (Przypadek 2)

Algorytm	Droga	Czas
FITSP	3.6259	02s
2-opt	2.4753	0.26s
3-opt	2.3126	0.97s
Sieć śr.	2.5147	4min 1.93s
Sieć min.	2.3999	4min 26.83s
Minimal.	2.2818	

Przykład 1 miał parametry przyjęte wg [Aiye90], przykład 2 ma przede wszystkim 8 razy większą wartość D niż przyjmowana w przykładzie 1. W przykładzie 1 każde z rozwiązań było

dopuszczalne, w przykładzie 2 tylko 8 spośród serii 15 rozwiązań miało przynajmniej jedno rozwiązanie dopuszczalne. Ilustruje to pewną ogólną własność optymalizacji za pomocą sieci neuronowych: Rozwiązania sieci TSP w przykładzie 1 są zawsze dopuszczalne ale są wyraźnie gorsze od uzyskiwanych na innej drodze (na przykład algorytmem 3-optymalnym). Rozwiązania sieci TSP w przykładzie 2 są znacznie lepsze od pozostałych, jednak jest to okupione faktem, że prawie połowa prób dostarczyła rozwiązań niedopuszczalnych.

Rozdział 11

Zakończenie

Książka, której lekturę właśnie kończymy, wprowadza problematykę sieci neuronowych w sposób uproszczony. Ograniczona objętość podręcznika wraz z nieslychanym bogactwem wiadomości, jakie dziś są dostępne na temat sieci (wystarczy porównać zamieszczoną bibliografię przedmiotu) spowodowały, że wyczerpująco omówiono jedynie najważniejsze, podstawowe zagadnienia, pozostawiając ogromną liczbę szczegółów samodzielnym studiom Czytelników. Studia te mogą być odbywane samodzielnie, na podstawie licznych dalszych książek i publikacji (m.in. wymienionych w spisie bibliografii), ale mogą być również prowadzone systemem akademickim w wybranych zagranicznych Uniwersytetach[1]. Ponieważ coraz latwiej jest ostatnio uzyskać środki na studia za granicą (m.in. z różnych międzynarodowych fundacji oferujących stypendia), warto może wiedzieć, gdzie się można udać po to, by głębiej i dokładniej studiować omawiane w książce zagadnienia. Niżej wymieniono (zebrane na podstawie sondaży obejmujących prawie całą kulę ziemską) te uczelnie, na których można studiować problematykę sieci neuronowych. Dla oszczędności miejsca (tradycyjny pocztowy adres jest bardzo długi) i dla zapewnienia Czytelnikom maksymalnej wygody, osiąganej dzięki stosowaniu środków nowoczesnej teleinformatyki, w wykazie podano jedynie adresy elektroniczej poczty (EMAIL), pod którymi można uzyskać konkretne informacje za pomocą dowolnego komputera pracującego w sieciach INTERNET, EARN albo BITNET. Na ile autrowi wiadomo, wszystkie szkoły wyższe w Polsce posiadają już dostęp do tych sieci, a praktyczne treningi w użytkowaniu elektronicznej poczty niewątpliwie każdemu się przydadzą!

Oto alfabetycznie uporządkowany wykaz interesujących Uczelni:

ABO AKADEMI UNIVERSITY, FINLAND
vt_ai@finabo.abo.fi

BIRMINGHAM UNIVERSITY
M.J.G.Harris@birmingham.ac.uk

BOSTON UNIVERSITY
caroly@park.bu.edu

[1]Niestety, w Polsce jedynie na studiach doktoranckich w Instytucie Automatyki AGH prowadzone są obszerne wykłady i wyczerpujące ćwiczenia laboratoryjne z zakresu sieci neuronowych. Na innych polskich uczelniach zagadnienia te nie są wykładane lub dostępne są jedynie jednosemestralne wykłady monograficzne, nie wykraczające zazwyczaj poza zakres tej książki i nie wnoszące wiele dla osób, które już ją przeczytały.

CALIFORNIA INSTITUTE OF TECHNOLOGY
jbower@smaug.cns.caltech.edu

CAMEBRIDGE
mavis@eng.cam.ac.uk

CHICAGO, UNIVERSITY OF
info@cs.uchicago.edu

COLORADO, UNIVERSITY OF AT BOULDER
gmcclella@clipr.colorado.edu

EDINBURGH, UNIVERSITY OF
betty@uk.ac.ed.cogsci

GEORGIA TECH
billman@pravda.cc.gatech.edu

GEORGIA, UNIVERSITY OF
mcovingt@uga.cc.uga.edu

INDIANA
barnesc@ucs.indiana.edu

GRENOBLE, UNIVERSITY OF
bessiere@imag.imag.fr

IOWA STATE
stolfus@judy.cs.iastate.edu

NEW HAMPSHIRE, UNIVERSITY OF
rmt@cs.unh.edu

NORTH CAROLINA
marshall@cs.unc.edu

OHIO_STATE
phil+@osu.edu

THE UNIVERSITY OF PITTSBURGH
neurocog@pittvms.bitnet.

POLYTECHNIC OF CENTRAL LONDON
konstan@uk.ac.pcl.mole

QUEENSLAND, UNIVERSITY OF
igsh@psych.psy.uq.oz.au

SUSSEX UNIVERSITY
andycl@uk.ac.sussex.syma

SYRACUSE UNIVERSITY
curt@cassi.cog.syr.edu

UCLA
verra@cs.ucla.edu

UCSD
swaney@cogsci.ucsd.edu

Podana wyżej propozycja mogła głównie interesować studentów poznających podstawy budowy i wykorzystania sieci neuronowych. Dla badaczy i naukowców bardziej interesująca okaże się zapewne informacja o międzynarodowym banku testowych problemów *(benchmarks)*, za pomocą których uzyskuje się możliwość porównań funkcjonowania sieci tworzonych przez różnych badaczy (np. szybkość uczenia, stopień trafności rozwiązywania problemów, wymagana moc obliczeniowa itp.). Dostęp do banku uzyskać można również w CMU za pomocą sieci INTERNET (na przykład programem **FTP**) pod adresem sieciowym **cs.cmu.edu**
w katalogu
/afs/cs.cmu.edu/project/connect/bench

Każde zadanie w banku ma postać pliku tekstowego, zawierającego następujące pola:

NAME: nazwa testu,

SUMMARY: krótki opis,

SOURCE: autor testu,

MAINTAINER: osoba w CMU opiekująca się testem,

PROBLEM DESCRIPTION: opis zadania stanowiącego test, zawierający tekst programu realizującego zadanie (zwykle w języku C lub LISP), oraz ciąg uczący w postaci opisu reguły lub konkretnego zbioru danych.

METHODOLOGY: opis sposobu realizacji testu,

VARIATIONS: opisy ewentualnych odmian testu,

RESULTS: zapis wyników uzyskanych przez różnych badaczy wykorzystujących test,

REFERENCES: odnośniki do opublikowanych prac różnych badaczy,

COMMENTS: uwagi badaczy wykorzystujących testy.

Opisana próba unifikacji badań sieci i ich wyników ma zapewne liczne wady, jednak sam fakt, że podjęto próbę takiej unifikacji, należy ocenić zdecydowanie pozytywnie.